Hanna-Kaisa Alanne

Standardoidut teknologiat verkkosovellusten modulaariseen kehittämiseen

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Mediatekniikan koulutusohjelma

Insinöörityö

30.4.2017

Metropolia

Tekijä Otsikko Sivumäärä Aika	Hanna-Kaisa Alanne Standardoidut teknologiat verkkosovellusten modulaariseen kehittämiseen 60 sivua 30.4.2017
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Mediatekniikka
Suuntautumisvaihtoehto	Digitaalinen media
Ohjaaja	Yliopettaja Harri Airaksinen

Insinöörityön tarkoituksena oli tutkia uuden teknologian, webkomponenttien, käyttöä verkkosovellusten modulaarisessa kehittämisessä ja toteuttamisessa. Työssä haluttiin selvittää, mitä uusia ulottuvuuksia webkomponentit tuovat sovelluskehitykseen ja mitä yleisiä webkehityksessä esiintyviä ongelmia ne voivat ratkaista. Lisäksi työssä perehdyttiin kahteen erilaiseen webkomponenttikirjastoon, joiden ominaisuuksia vertailtiin tarkoituksena selvittää, kumpi soveltuisi paremmin verkkosovelluksen toteuttamiseen.

Webkomponenttikirjastoista vertailuun valittiin Mozillan X-Tag ja Googlen Polymer. Molemmilla kirjastoilla toteutettiin pienet esimerkit, ja vertailun lopputuloksena valittiin Googlen Polymer varsinaisen työn toteutukseen. Koska webkomponenttien tavoitteena on tuoda muun muassa modulaarisuutta verkkosovelluksiin, toteutetussa sovelluksessa oleellista oli sen modulaarinen rakenne ja toiminnallisuuksien jakaminen selkeiksi erillisiksi uudelleen käytettäviksi komponenteiksi.

Toteutettu sovellus hyödynsi webkomponenttiteknologioita, joiden avulla merkintäkieli (HTML), komentosarjakieli (JavaScript) ja tyylit (CSS) saatiin tiivistettyä erillisiksi uudelleen käytettäviksi komponenteiksi. Työssä huomattiin webkomponenttien ja Polymerin käytössä sekä hyviä että huonoja puolia. Toiminnallisuuksien ja tyylien kapseloiminen uudelleen käytettäviksi komponenteiksi vähensi modulaarisuuteen liittyviä ongelmia, mutta sivun latausaika liitännäisten kanssa kasvoi melko suureksi. Tulevaisuudessa tärkeä kehityskohde on webkomponenttien tuen saaminen kaikkiin moderneihin selaimiin.

Insinöörityön perusteella webkomponentit tuovat modulaarisen verkkosovelluksen kehitykseen hyviä ratkaisuja. Ne tarjoavat uusia mahdollisuuksia toteuttaa sovelluksia erillisistä uudelleen käytettävistä komponenteista, joiden toiminta on muista komponenteista riippumatonta. Webkomponentit voivat yleisesti kasvattaa suosiotaan lähivuosina, jos niiden tuki saadaan lisättyä kaikkiin selaimiin.

Avainsanat	webkomponentit, verkkosovellus, Polymer, ohjelmointirajapinta, modulaarisuus

Author	Hanna-Kaisa Alanne
Title	Standardized technologies for modular web application development
Number of Pages	60 pages
Date	30 April 2017
Degree	Bachelor of Engineering
Degree Programme	Media Technology
Specialisation option	Digital Media
Instructor	Harri Airaksinen, Principal Lecturer

The purpose of this thesis was to study a new technology called web components, and their role in designing and developing modular web applications. In this project the intention was to find out what new possibilities web components can bring to application development, and which common problems they can resolve. In addition, this thesis focused on two different web component libraries, comparing their features, trying to find out which library was better for implementing a web application.

From web component libraries Mozilla's X-Tag and Google's Polymer were chosen to comparison. I created small examples with both libraries. After comparing these two libraries and their features, I chose to use Google's Polymer to implement the actual project. Because web components aim to more modular web applications, it was essential in my application that it had modular structure and that all the functions were divided into separate reusable components.

The application was made using web component technologies which allow developers to encapsulate markup (HTML), scripts (JavaScript) and stylesheets (CSS) into reusable packages. The findings of this study indicate that there are pros and cons when using web components and Polymer. Encapsulating functions and stylesheets into reusable components reduced the common problems which appear with modular development, but because of large polyfills it also increased page download time a lot. In the future, it is important that web components will be implemented in all browsers so that there is no need for polyfills.

In conclusion, web components can bring good solutions to modular web application design and development. Web components offer new possibilities to develop applications using reusable components that work without conflicting with any other parts. Web components can become very popular in the future, if their support will be implemented in all modern browsers.

Keywords	web components, web application, Polymer, application programming interface, modularity

Sisällys

1 Johdanto

2000-luvun alkupuolella internetin suosion kasvaessa ja teknologioiden kehittyessä alettiin saada käsitystä siitä, mitä mahdollisuuksia internet voisi tuoda yrityksille. Internetiä alettiin käyttää kustannustehokkaana kommunikaatiovälineenä yritysten ja kuluttajien välillä. Uusia sivustoja alkoi ilmestyä nopeasti, ja pian kehittäjät alkoivat pohtia, kuinka verkkosivuista saataisiin vieläkin tehokkaampia tiedonsiirtovälineitä. Verkkosivuille alkoi ilmestyä elementtejä, jotka mahdollistivat interaktiivisen kommunikoinnin käyttäjän ja verkkosivuston välillä. Vuosien kuluessa alkuaikojen staattiset verkkosivut ovat muuntuneet dynaamisiksi verkkosovelluksiksi.

World Wide Web Consortium, lyhyemmin W3C, on määritellyt verkkosovelluksen verkkosivustoksi, joka käyttää asiakasohjelman ja palvelimen väliseen tiedonsiirtoon http-protokollaa ja jossa tieto prosessoidaan joko palvelimella tai asiakasohjelmassa ja näin luodaan sisäänrakennetun (native) sovelluksen kaltainen käyttäjäkokemus selaimen kautta [1]. Verkkosovelluksien palvelinpuolen ohjelmointikielistä suosituin on PHP [2]. Asiakaspuolen ohjelmointikielistä ylivoimaisesti käytetyin on JavaScript. Jopa 94 % verkkosivuista on toteutettu JavaScriptiä hyödyntäen [3]. JavaScriptin kehitys toi mukanaan mahdollisuuden lisätä dynaamisuutta verkkosivuille, ja sen käyttö teki verkkosivujen sisällön lataamisesta nopeampaa [4]. Yksinkertaisen JavaScriptin syntaksi on kuitenkin monimutkaista, ja siksi sen tueksi on vuosien varrella kehitetty satoja kirjastoja ja ohjelmistokehyksiä, joiden tarkoituksena on helpottaa ja yksinkertaistaa JavaScriptin käyttöä.

JavaScriptin lisäosat ja komponentit ovat rikkonaisia kokonaisuuksia, ja ne ovat riippuvaisia kirjastoista ja ohjelmistokehyksistä, jotka eivät välttämättä ole yhteensopivia toistensa kanssa. Webkomponentit pyrkivät tuomaan ratkaisun tähän ongelmaan. Webkomponentit ovat tapa standardoida sovelluksessa käytettäviä komponentteja ja luoda uudelleenkäytettäviä lisäosia. Webkomponenttien tarkoituksena on tuoda verkkosovelluskehitykseen komponenttipohjaisuutta ja modulaarisuutta, ja niiden etuna on, että ne ovat yhteensopivia kaikkien kirjastojen ja ohjelmistokehysten kanssa. [5.]

Webkomponentit ovat tuore käsite verkkosovelluskehityksessä, ja niihin liittyy toistaiseksi rajoitteita. Isoin ongelma on, että kaikki verkkoselaimet eivät vielä tue webkomponenttien käyttöä kokonaan. Ongelman ratkaisuksi on kehitetty kirjastoja, jotka sisältävät liitännäisiä (polyfills), joiden avulla webkomponenttien tuki voidaan saada kaikkiin

moderneihin selaimiin. Tunnetuimmat webkomponenttikirjastot ovat Googlen Polymer ja Microsoftin X-Tag. [5.]

Insinöörityön tarkoituksena on selvittää, miten ja miksi webkomponentteja käytetään verkkosovellusten kehittämisessä. Tarkemmin perehdytään Googlen kehittämän Polymer-kirjaston toimintaan ja sen rooliin webkomponenttien käytössä. Insinöörityössä perehdytään myös X-Tagin ominaisuuksiin ja verrataan niitä Polymerin ominaisuuksiin. Tarkoituksena on selvittää, mitä uutta webkomponentit tuovat verkkosovelluskehitykseen ja miten niiden käyttö helpottaa kehitystyötä.

Insinöörityössä vertaillaan webkomponenttien käyttöä helpottavien kirjastojen ominaisuuksia pienten esimerkkien avulla. Isompi sovellus toteutetaan Polymer-kirjastoa hyödyntäen ilman mitään muuta kirjastoa tai ohjelmistokehystä. Sovelluksen tarkoituksena on esitellä webkomponenttien käyttöä ja antaa osviittaa siitä, miten webkomponentteja voi hyödyntää sovelluksissa.

2 Verkkosovellusten kehitys

2.1 Verkkosovellusten historiaa ja arkkitehtuuri

Internet, nykymaailman työväline tiedonsiirtoon ja kommunikoimiseen, on kehittynyt viimeisten 20 vuoden aikana paljon. Internet kehitettiin jo kauan ennen sen maailmanlaajuista leviämistä, mutta vasta 1990-luvun alkupuolella, kun teknologia oli tarpeeksi kehittynyttä, internet kaupallistettiin. Ensimmäinen internetselain Mosaic julkaistiin vuonna 1993, ja vuotta myöhemmin internet levisi suuren yleisön tietoisuuteen kaupallistumisen myötä. Kun ihmiset ymmärsivät internetin tuomat mahdollisuudet, verkkosivuja alkoi ilmestyä nopeasti. [6.]

Alkuaikojen verkkosivut oli tarkoitettu lähinnä tiedonsiirtoon, ja ne oli toteutettu HTML:n, eli HyperText Markup Languagen, avulla. Ensimmäiset verkkosivut olivat toiminallisuuksiltaan yksinkertaisia staattisia sivuja, joiden päätarkoituksena oli välittää tietoa käyttäjille. Verkkosivuihin haluttiin kuitenkin lisätä dynaamisuutta, mutta HTML ei soveltunut yksinään monimutkaisempien sivujen ja sovellusten kehittämiseen rajallisten toimintojensa vuoksi. Internetiä haluttiin hyödyntää enemmän, ja sitä haluttiin käyttää monimutkaisimpien sovellusten toteuttamiseen, joten uusien teknologioiden oli synnyttävä

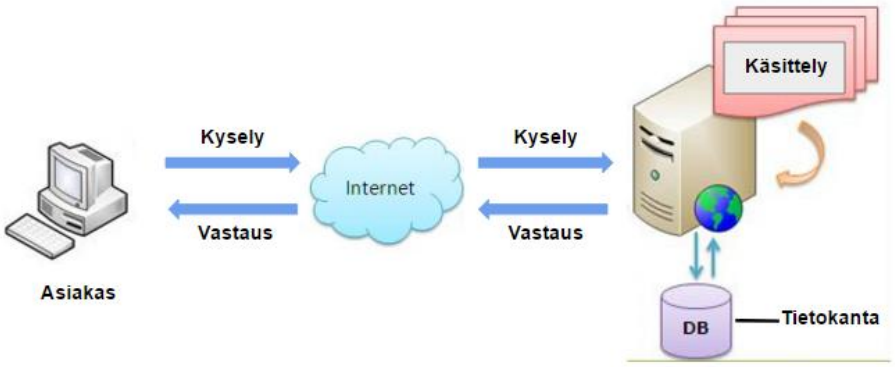
HTML:n rinnalle. Vuonna 1995 Netscape Communications esitteli JavaScriptin, dynaamisen komentosarjakielen, jonka avulla verkkosivuille pystytään lisäämään dynaamisia toiminnallisuuksia. JavaScriptin lisäksi muita teknologioita, kuten PHP, ASP ja Java, ja myöhemmin myös Ajax, kehitettiin monimutkaisempien verkkosivujen toteuttamiseen. Nämä teknologiat mahdollistavat nykypäivänäkin dynaamisten verkkosivujen, niin sanottujen verkkosovellusten, toteuttamisen. [6.]

Verkkosivun ja verkkosovelluksen käsitteet sekoittuvat usein keskenään. Verkkosivu käsitteenä on juurtunut ihmisten mieliin, ja siksi sitä käytetäänkin usein myös verkkosovelluksista puhuttaessa. Kuitenkin tietotekniikan ammattilaisilta kysyttäessä verkkosivujen ja verkkosovellusten ero on usein selvästi määritelty. Verkkosivujen tarkoituksena on tuottaa informaatiota käyttäjille, kun taas verkkosovellukset ovat vuorovaikutteisia käyttäjien kanssa. Voidaan sanoa, että verkkosivut on toteutettu vain lukemista varten ja verkkosovelluksissa käyttäjä voi sekä lukea että tuottaa sisältöä. Verkkosovellus on itsenäinen sisäänrakennettua sovellusta muistuttava, sisällöltään interaktiivinen ja toimintasuuntautunut sivusto, joka toimii selaimen kautta. Sovellus voi hyödyntää laitteiden, kuten älypuhelimien ja taulutietokoneiden, ominaisuuksia, esimerkiksi paikannusta ja kameraa. [7.]

Sovelluskehittäjät ovat nykyään keskittyneet kehittämään natiivisovellusten lisäksi verkkosovelluksia. Verkkosovellusten suurin etu natiivisovelluksiin verrattuna on niiden laiteriippumattomuus. Juuri laiteriippumattomuus selittää verkkosovellusten suosion kasvua kehittäjien ja käyttäjien keskuudessa. Käyttäjien ei tarvitse asentaa sovellusta laitteelleen, vaan he pääsevät käyttämään sovellusta internetin ja selaimensa välityksellä. Laiteriippumattomuus on myös suuri etu kehittäjille. Heidän ei tarvitse miettiä, mille laitteelle ja käyttöjärjestelmälle he haluavat sovelluksen kehittää, vaan sovellus toimii kaikissa laitteissa ja käyttöjärjestelmissä. HTML5:sen ja selainten sovellusrajapintojen myötä kehittäjien on ollut mahdollista luoda verkkosovelluksilla käyttäjille samankaltaisia käyttäjäkokemuksia kuin natiivisovelluksillakin. Nämä tekniikat ovat mahdollistaneet niin sanottujen yksisivuisten sovellusten toteuttamisen, jolloin sovelluksiin on saatu natiivisovellusten kaltaisia toiminnallisuuksia. [8.]

Verkkosivujen ja verkkosovellusten toiminta perustuu verkkopalvelimiin ja asiakkaina toimiviin tietokoneisiin ja selaimiin, jotka lähettävät pyyntöjä palvelimille. Kuvassa 1 on kuvattu yksinkertaistettuna verkkosovelluksen toiminta. Asiakasohjelma, usein käyttäjän

käyttämä selain, lähettää palvelimelle http-pyynnön, palvelin prosessoi pyynnön ja lähettää asiakasohjelmalle vastauksen, tavallisimmin HTML-tiedoston tai binääridataa. [9.]



Kuva 1. Yksinkertaisen verkkosovelluksen toiminta [muokattu lähteestä 10].

Palvelinpuoli

Asiakaspuolen ja palvelinpuolen toiminnot hoidetaan eri ohjelmointikielillä. Palvelinpuolen ohjelmointikielet hoitavat sovelluksen taustalla tapahtuvat toiminnot, kuten kommunikoinnin tietokannan kanssa, ja datan varastoimisen [9]. Palvelinpuolen ohjelmointikieliä ja ohjelmointiympäristöjä on useita, mutta suosituimpia ovat

- ASP.NET – Microsoftin webkehitykseen kehittämä kokoelma työkaluja, joiden avulla voidaan tehdä dynaamisia verkkosivuja HTML:n, CSS:n, JavaScriptin ja eri komentosarjakielien avulla [11; 12]
- ColdFusion – verkkosovellusten kehittämisympäristö ja ohjelmointikieli tehokkaiden sovellusten toteuttamiseen [13]
- Java – Sun Microsystemsin kehittämä olio-ohjelmointikieli modulaaristen ohjelmien luomiseen [14]
- PHP – suosituin palvelinpuolen avoimen lähdekoodin ohjelmointikieli, joka on erityisesti tarkoitettu webkehitykseen; PHP:n avulla on helppo luoda dynaamisia verkkosovelluksia [11; 15].

Asiakaspuoli

Asiakkaana toimii käytettävä tietokone tai sovellus, joka lähettää pyyntöjä palvelinpuolelle. Asiakaspuolen ohjelmointikielet vastaavat siitä, miten sovellus näyttäytyy käyttäjälle, mistä elementeistä sovellus koostuu ja mitä interaktiivisuutta se sisältää. Käytetyimpiä asiakaspuolen tekniikoita ovat HTML ja CSS. Melkein kaikki verkkosivut sisältävät näitä kahta tekniikkaa [9]. Jotta sivustoon saadaan lisättyä dynaamisuutta, tarvitaan HTML:n ja CSS:n lisäksi komentosarjakieli. Käytetyimpiä asiakaspuolen tekniikoita ja ohjelmointikieliä on lueteltu seuraavassa listassa.

- CSS (Cascading Style Sheets) – Tyylijärjestelmä, joka määrittelee HTML-dokumentin tyylin ja sen, miten elementit esitetään käyttäjälle. CSS:n avulla muokataan verkkosovellusten ulkoasua [16]
- Flash – Kehitysympäristö multimediasovellusten luomiseen. Flash käyttää ActionScript-komentosarjakieltä [17]
- HTML (HyperText Markup Language) – Verkkosovellusten rakennukseen tarkoitettu merkintäkieli, joka koostuu elementeistä, jotka määrittelevät sivuston rakenteen [18]
- JavaScript – Suosituin komentosarjakieli, joka mahdollistaa käyttäjän toimintoihin reagoinnin, DOMin (Document Object Model, dokumenttiolio-malli) muokkaamisen ja palvelinpyyntöjen lähettämisen ja vastaanottamisen. Mahdollistaa verkkosovellusten dynaamiset toiminnot [19]
- TypeScript – Microsoftin kehittämä avoimen lähdekoodin komentosarjakieli [20].

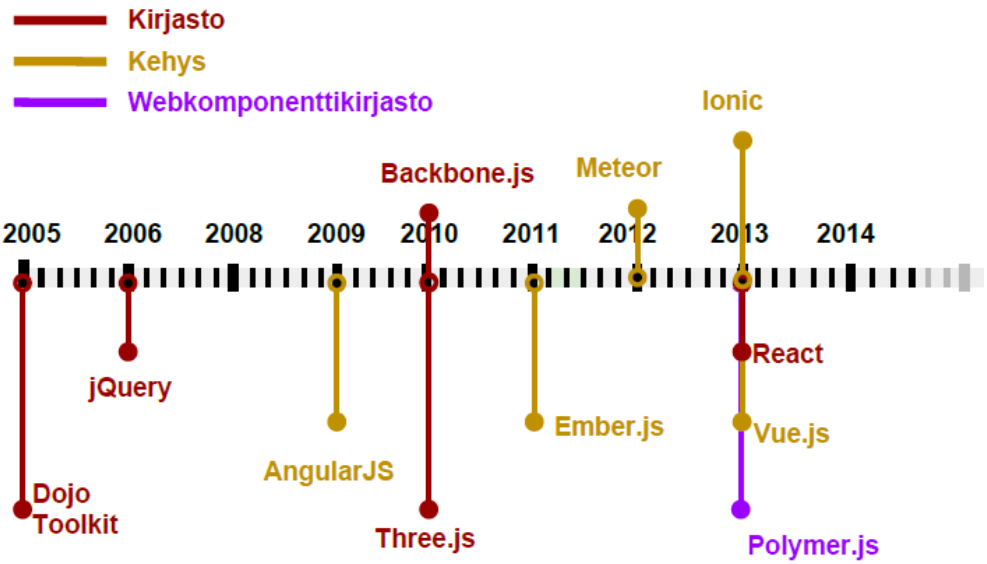
Aikaisemmin Flash ja ActionScript olivat suosittu tapa toteuttaa dynaamisia verkkosivuja, mutta HTML5:sen ja JavaScriptin suosio on ajanut niiden ohi. Nykyään JavaScript on maailmanlaajuisesti suosituin komentosarjakieli webkehityksessä [19]. Pelkän yksinkertaisen JavaScriptin, niin sanotun VanillaJs:n, käyttö kuitenkin tekee koodista monimutkaista, ja siksi sen tueksi on kehitetty satoja erilaisia kirjastoja ja ohjelmistokehyksiä.

2.2 JavaScriptin kirjastot ja ohjelmistokehykset

Viimeisten reilun kymmenen vuoden aikana JavaScriptin tueksi on kehitetty todella paljon erilaisia kirjastoja ja ohjelmistokehyksiä. Määrästä antaa osviittaa se, että GitHubista haettaessa hakusanoilla ”javascript library” tulokseksi saadaan 16 607 eri kirjastoa, ja hakusanoilla ”javascript framework” tuloksena on 6 797 eri ohjelmistokehystä [21; 22].

Yhteensä kehitettyjä kirjastoja ja ohjelmistokehyksiä voi olla yli 20 000. Näiden kirjastojen ja ohjelmistokehysten tarkoituksena on ollut tuoda jotain uutta ja mullistavaa sovelluskehitykseen. Vaihtoehtojen runsaudenpulan takia läheskään kaikki eivät kykene saavuttamaan suuren yleisön suosiota, ja siksi ne unohtuvat. Jotkin kirjastot ja ohjelmistokehykset ovat kuitenkin pystyneet vakuuttamaan kehittäjät, ja niiden suosio jatkuu vuodesta ja versiosta toiseen. Seuraavassa esitellään muutama suosituin JavaScript-kirjasto ja -ohjelmistokehys.

Kuvaan 2 on merkitty muutaman suosituimman JavaScript-kirjaston ja -ohjelmistokehyksen ensimmäisten versioiden julkaisuvuodet. Aikajana rajoittuu vuosiin 2005–2013, koska edelläkävijät ja tällä hetkellä suosituimmat kirjastot ovat ilmestyneet tällä välillä. Kirjastot on merkitty punaisella ja ohjelmistokehykset keltaisella. Lisäksi violetilla on merkitty Polymer-webkomponenttikirjasto.



Kuva 2. Muutaman suosituimman JavaScript-kirjaston ja -ohjelmistokehyksen ensimmäisten versioiden julkaisuvuodet [muokattu lähteistä 5; 23; 24].

Vuonna 2005 julkaistiin Dojo Toolkit, ja se toi käyttäjille mahdollisuuden lisätä verkkosivuille monimutkaisia elementtejä. Dojo Toolkit esitteli kehittäjille widgetit eli niin sanotut pienoisohjelmat. Käyttäjä pystyi lisäämään esimerkiksi kaavion verkkosivulleen kirjoittamalla vain muutaman rivin koodia. Dojon myötä kehittäjät alkoivat ymmärtää uudelleen käytettävien komponenttien hyötyjä. Dojo Toolkitia voidaan pitää modulaarisen suunnittelun ja webkomponenttien rakentamisen edelläkävijänä. [5.]

Reilu kymmenen vuotta sitten esiteltiin jQueryn ensimmäinen versio. Siitä lähtien kirjaston suosio on kasvanut, ja nyt se on käytetyin JavaScript-kirjasto. Jopa 77 prosenttia

verkossa olevista sivuista käyttää jossain määrin jQueryä [24]. jQuery mahdollistaa tiiviimmän koodin kirjoittamisen, ja sen avulla voidaan yksinkertaistaa monia JavaScriptin monimutkaisia ominaisuuksia. Pari vuotta myöhemmin julkaistiin jQueryUI, joka on kokoelma erilaisia käyttöliittymäkomponentteja. [5; 25; 26.]

AngularJS, jonka ensimmäinen versio julkaistiin vuonna 2009 ja ensimmäinen vakaa versio vuonna 2011, on yksi suosituimmista JavaScriptin ohjelmistokehyksistä. AngularJS sallii HTML:n syntaksin laajentamisen direktiiveillä, joiden ansiosta muun muassa kaksisuuntainen datakytkentä on helppo toteuttaa pienemmällä koodimäärällä. AngularJS on toteutettu MVC-arkkitehtuurin (Model-View-Controller) mukaisesti, ja se on kirjoitettu kokonaan JavaScriptillä. [23.]

Backbone.js:n ja Ember.js:n ensimmäiset versiot julkaistiin peräkkäisinä vuosina 2010 ja 2011, ja niitä voidaan pitää yksinä AngularJS:n suurimmista kilpailijoista, mutta ainakaan vielä ne eivät ole kyenneet haastamaan AngularJS:n suosiota ohjelmistokehysten välisessä kilpailussa. Ember.js:n tavoitteena on antaa käyttäjälle mahdollisuus keskittyä toimintojen ja ulkoasun toteuttamiseen sen hoitaessa taustalla kaikki sellaiset toiminnot, jotka ovat yleisiä kaikissa verkkosovelluksissa. Backbone.js on kevyt ohjelmistokehys, joka mahdollistaa MVC-arkkitehtuurin mukaisen rakenteen. Vuonna 2010 julkaistiin myös Three.js-kirjasto, jota käytetään 3D-grafiikan luomisessa verkkosovelluksiin. [23; 24.]

Vuonna 2012 julkaistu Meteor tarjoaa kehittäjille ohjelmistokehyksen, joka sisältää kaikki toiminnot, joita sovellusten kehittämiseen tarvitaan: käyttöliittymän esittämisen, palvelinpuolen toiminnot ja tietokannan hallinnan. Vuonna 2013 julkaistiin Ionic- ja Vue.js-ohjelmistokehykset ja React.js-kirjasto. Ionic tarjoaa ohjelmistokehyksen mobiilisovellusten kehittämiseen ja Vue.js interaktiivisten sovellusten kehittämiseen. Vue.js toimii MVVM-arkkitehtuurin (Model-View-View-Model) periaatteella. Myös React.js- ja Polymer.js-kirjastot julkaistiin vuonna 2013. React.js tarjoaa kehittäjille ohjelmistokehyksen, joka on keskittynyt käyttöliittymien toteuttamiseen. Polymer.js on webkomponenttikirjasto, jonka avulla kehittäjät voivat rakentaa ja uudelleen käyttää olemassa olevia webkomponentteja. [23; 24.]

Niin kuin luvun alussa mainittiin, JavaScriptin tueksi on kehitetty jopa 20 000 erilaista kirjastoa ja ohjelmistokehystä, joten tässä luvussa on kerrottu vain hyvin pienestä murto-

osasta. Olemassa olevien kirjastojen lisäksi koko ajan kehitetään uusia vaihtoehtoja tuomaan ratkaisuja eri ongelmiin. Myös olemassa olevista kirjastoista ja ohjelmistokehyksistä pyritään kehittämään parempia, ja niistä julkaistaan uusia versioita aika ajoin. Hyvänä esimerkkinä on AngularJS:n seuraaja Angular2, jonka tavoitteena on ollut tuoda kaivattuja ominaisuuksia ohjelmistokehykseen ja tehdä verkkosovellusten kehittämisestä entistä helpompaa ja sujuvampaa.

Viime vuosina modulaarisuus ja erillisten uudelleen käytettävien komponenttien merkitys on kasvanut webkehityksessä, ja siksi myös tekniikoiden kehittyessä webkomponenttikirjastoja on alettu kehittää. Tästä esimerkkinä on Googlen Polymer.js-kirjasto, joka tuo kehittäjille mahdollisuuden luoda komponentteja helposti. Seuraavissa luvuissa keskitytään modulaarisuuteen verkkosovelluskehityksessä sekä webkomponentteihin ja webkomponenttikirjastoihin ja niiden käyttöön.

3 Webkomponentit verkkosovelluksien toteutuksessa

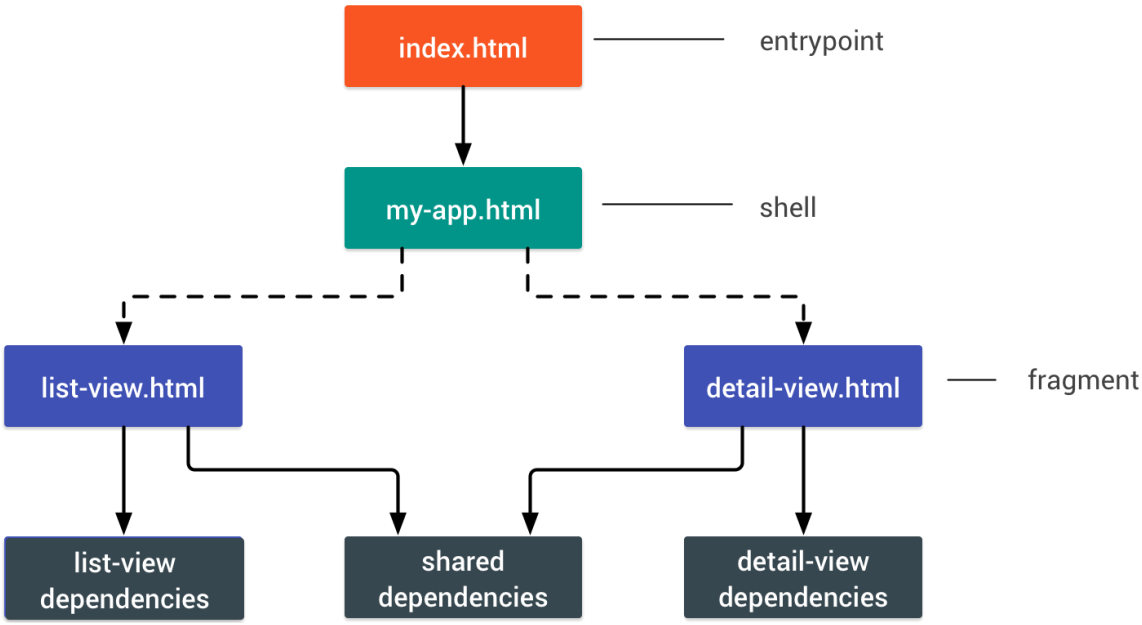
3.1 Modulaarisuus ja komponentit

Nykypäivän verkkosovelluskehityksessä modulaarisuuden suosio on kasvanut. Modulaarisuuden tarkoituksena on rakentaa verkkosovelluksen lähdekoodi erillisistä komponenteista, jotka ovat uudelleen käytettäviä ja laajennettavissa. Modulaarisuuteen pohjautuva suunnittelu luo sovellukselle rakennetta ja lisää tehokkuutta. Onnistuneessa modulaarisessa toteutuksessa komponenttien oletetaan toimivan ilman konflikteja toisten osien kanssa. Tapa, jolla sovelluksen koodi on jäsennelty ja komponenttien saumaton toiminta keskenään, on ratkaisevassa asemassa sovelluksen optimaalisen toiminnan kannalta. Modulaarisuus ja erillisten komponenttien luominen on kasvattanut suosiotaan, ja se tuo monia etuja selainpuolen kehitykseen. [27.]

Komponentti määritellään usein kokonaisuuden osaksi tai osatekijäksi. Front-end -kehityksessä komponentti määritellään itsenäiseksi, selkeästi määritellyksi, uudelleen käytettäväksi kokonaisuuden osaksi, joka sisältää sille ominaiset toiminnallisuudet. Itsenäistä komponenttia tulisi voida käyttää omillaan, ja ne pitäisi rakentaa ilman, että ne vaikuttavat muiden komponenttien toimintaan. Hyvin suunniteltu ja toteutettu kompo-

nentti sisältää vain sille ominaiset toiminnallisuudet ja muut ominaisuudet, ja se on uudelleen käytettävissä. Komponentit tuovat koodiin johdonmukaisuutta, ja niitä pystyy nopeasti päivittämään tarvittaessa. [28.]

Kuvassa 3 on esitetty esimerkki verkkosovelluksen arkkitehtuurista, joka on toteutettu modulaarisella tavalla erillisistä komponenteista. Päätasona toimii index.html, ja sovelluksen toimintalogiikat, kuten reititys, on sisällytetty my-app.html-elementtiin. Sovelluksen jokainen näkymä on luotu eri elementteihin (list-view.html ja detail-view.html), jotka ladataan sitä mukaa, kuin käyttäjä niihin navigoi. Elementteihin voi sisältyä joitain elementtikohtaisia riippuvuuksia ja myös yhteisiä riippuvuuksia. Tällainen arkkitehtuuri muun muassa minimoi sivuston latausnopeutta, koska eri näkymät latautuvat vasta käyttäjän toimintojen perusteella. [29.]



Kuva 3. Modulaarisen verkkosovelluksen arkkitehtuuri [29].

Modulaarinen suunnittelu ja komponenttien käyttö voi kuitenkin aiheuttaa muutamia ongelmia kehityksessä. Esimerkiksi osatekijöiden erottelun pitäisi perustua siihen, että jokainen osa keskittyy vain yhden asian toteuttamiseen. Usein kuitenkin, varsinkin isoissa projekteissa, modulaarisuus voi osoittautua hankalaksi, ja se saattaa johtaa HTML:n, CSS:n ja JavaScriptin liian tiukkaan yhteen kytkemiseen (coupling). Tämä taas saattaa aiheuttaa sen, että koodiin tehtävät muutokset aiheuttavat jonkin muun osan toimimattomuuden. Toinen iso ongelma on CSS:n globaalit valitsimet, koska jokainen tyylisääntö voi vaikuttaa johonkin aivan erilliseen osaan. HTML:n ongelmana on, että olemassa olevia elementtejä ei voi muokata tai laajentaa. Näiden ongelmien välttämiseksi ja modulaarisuuden helpottamiseksi on kehitetty useita kirjastoja ja ohjelmistokehyksiä. [27.]

Modulaarisuutta ja komponenttipohjaista arkkitehtuuria on pyritty tuomaan verkkosovelluskehitykseen jo monien kirjastojen ja ohjelmistokehyksien avulla. Esimerkiksi Front-end-ohjelmistokehykset Bootstrap ja Foundation ovat kehittäneet uudelleen käytettäviä komponentteja käyttöliittymien muokkaamiseen. JavaScript-kirjastoista ja -ohjelmistokehyksistä React.js ja AngularJS ovat mahdollistaneet koodin järjestämisen järkeviin osiin ja käyttöliittymäelementtien uudelleen käytettävyyden. Esimerkiksi AngularJS:n direktiivit ovat hyviä esimerkkejä modulaarisuudesta. Direktiiveissä määritellään halutut toiminnallisuudet, jotka liitetään tiettyyn DOM-elementtiin [30]. jQueryn UI-kirjasto toi käyttäjille kokoelman erillisiä käyttöliittymäkomponentteja. Vaikka kirjastoilla ja ohjelmistokehyksillä on pyritty välttämään ja kiertämään modulaarisuuden aiheuttamia ongelmia, ei niiltä aina pystytä välttymään. Standardoitu ratkaisu, joka määrittäisi tavan kehittää uudelleen käytettäviä, natiivisti laajennettavia ja tiiviitä komponentteja, olisi optimaalinen ratkaisu modulaarisuuden ongelmiin. Tähän tarkoitukseen on alettu kehittää webkomponentteja, joiden tavoitteena on standardien tuominen komponenttien luomiseen ja käyttöön. [27; 28.]

Alun perin webkomponentti-termiä käytti Microsoft kuvatessaan Officen lisäosia, ja vuosien varrella myös muut yritykset ovat käyttäneet webkomponentti-termiä eri asiayhteyksissä. Kuitenkin vuoden 2011 alusta lähtien webkomponentti-termillä on viitattu W3C-standardiin. Voidaan ajatella, että modernien webkomponenttien kehitys alkoi jo vuonna 2005, kun Ajax mahdollisti asynkronisen tiedon hakemisen palvelimelta ja DOJO Toolkit esitteli lisäosien lisäämisen verkkosivuille. Luvussa 2 esitelty jQuery toi mukanaan ominaisuuksia (”komponentteja”), joiden avulla JavaScript-koodia voitiin yksinkertaistaa ja DOMin manipulointia helpottaa. Vuoden 2010 tienoilla asiakaspuolen kehitys (client-side) ja yksisivuisten sovellusten suosio johti MV*-suunnittelumallin syntyyn. MV*-suunnittelumallissa kirjain M viittaa malliin (model) ja V näkymään (view). *-merkki viittaa erikseen määriteltävään osaan suunnittelumallissa. Suunnittelumalleja on useita, ja muun muassa MVC-suunnittelumallin C-kirjain viittaa ohjaimeen (controller). MV*-suunnittelumallien kohdalla uudelleen käytettävien ja kapseloitujen käyttöliittymäkomponenttien tärkeys kasvoi suureksi yksisivuisten sovellusten ja ohjelmistokehysten koodin määrän kasvaessa. [27; 31.]

JavaScriptin komponenttipohjaisiin kirjastoihin ja ohjelmistokehyksiin sisältyy rajoitteita, joihin webkomponentit pyrkivät tuomaan ratkaisuja. Yhtenä isona ongelmana voidaan pitää sitä, että eri kirjastojen ja ohjelmistokehysten komponentteja ei voi yhdistellä hel-

posti keskenään ilman konflikteja. Toinen ongelma komponenttipohjaisessa toteutuksessa ovat selainalustat ja niiden valmiudet tukea eri ominaisuuksia. Webkomponenttien ratkaisuna on tarjota natiivi selaintuki käyttöliittymäkomponenttien laajentamiselle, kapseloinnille ja niiden liittämiselle koodiin. Kapselointi ja uudelleen käytettävyys mahdollistavat komponenttien käytön minkä tahansa kirjaston ja ohjelmistokehyksen kanssa. [27.]

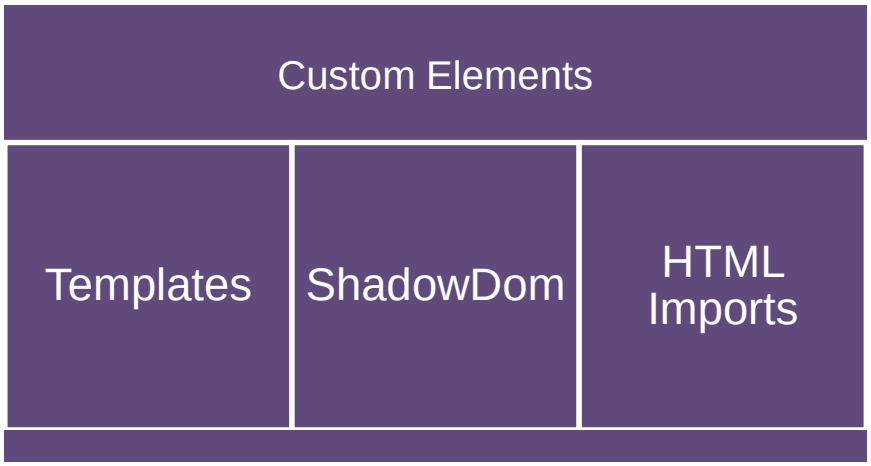
Webkomponentteihin liittyy useita eri tekniikoita, ja koska standardi on yhä kehitteillä, myös yleisiä ongelmia esiintyy. Suurin ongelma webkomponenttien käytössä on selaintuen puute. Vain pieni osa selaimista tukee kaikkia webkomponenttiteknologioita. [27.] Seuraavassa luvussa esitellään webkomponenttien ominaisuuksia, toimintaa, ja niihin liittyviä ongelmia.

3.2 Webkomponentit

Webkomponentit ovat kokoelma W3C:n kehitteillä olevia standardeja, jotka mahdollistavat merkintäkielen, komentosarjakielen ja tyylien tiivistämisen uudelleenkäytettäviksi paketeiksi. Webkomponentit koostuvat neljästä teknologiasta, jotka toimivat osana käyttäjän selainta, jolloin erillisiä kirjastoja ei tarvita. Webkomponenttien avulla verkkosovelluksista saadaan rakennettua modulaarisia, ja omien komponenttien luominen on helppoa. [27.]

Jokaisella HTML-elementillä, kuten esimerkiksi <button>-elementillä, on määritelty ohjelmointirajapinta ja tuki rakennettuna internetselaimeen. Selaimen tehtävänä on hakea elementin tiedot koodista, esittää elementti sivulla ja reagoida käyttäjän ja JavaScriptin määrittämiin toimintoihin. Webkomponenttien avulla kehittäjät voivat rakentaa omia HTML-elementtejä, jotka mahdollistavat merkintäkielen (HTML), komentosarjakielen ja tyylien kapseloimisen uudelleen käytettäviksi paketeiksi, joita selaimet tukevat natiivisti. [27.]

Webkomponentit koostuvat neljästä teknologiasta (kuva 4), W3C:n luomasta niin sanotusta suosituksesta. Nämä teknologiat ovat HTML-sivupohjat (templates), räätälöidyt elementit (custom elements), varjo-DOM (ShadowDOM) ja tuonti-HTML (HTML import). [32.]



Kuva 4. Webkomponentit. Muokattu Code Teddyn mallista [32].

Sivupohjat (Templates)

Sivupohja (template) luo niin sanotusti pohjapiirustuksen uudelle komponentille. Webkomponenteissa sivupohja luodaan <template>-elementin sisälle. Sivupohja voi sisältää sille ominaisen HTML-koodin ja tyylit, ja se voidaan myöhemmin kloonata ja uudelleen käyttää. Koodiesimerkeissä 1 ja 2 on esimerkki sivupohjan luomisesta henkilön profiilitietoja sisältävälle elementille. Sivupohja luodaan <template>- ja </template>-merkkien (tag) sisälle. Sivupohja voidaan sijoittaa mihin tahansa kohtaan koodissa. Koodiesimerkeissä luodaan <template>-elementti, joka sisältää henkilön tietoja. Ensin määritellään elementin sisältö koodiesimerkissä 1. [27; 33.]

```
1. <template id=”profileTemplate”>
2.   <div class=”profile”>
3.     <img src=”” class=”profile__img”>
4.     <div class=”profile__name”></div>
5.     <div class=”profile__social”></div>
6.   </div>
7. </template>
```

Koodiesimerkki 1. Template-elementin määritys [34].

Elementille asetetaan id-tunniste, ja <template>-elementin sisällä määritellään halutut elementin tiedot. Tässä tapauksessa elementti sisältää luokan ”profile”, jossa luodaan luokat ”profile__img”, ”profile__name” ja ”profile_social”.

Koodiesimerkissä 2 <template>-elementti aktivoidaan JavaScriptillä käyttämällä document.querySelector()-funktiota. Profiilin kuva, nimi ja sosiaalinen media määritetään tässä osiossa, ja sisältö lisätään <body>-elementin sisälle document.body.appendChild()-funktiota käyttämällä.

```
1. var template = document.querySelector('#profileTemplate');
2. template.querySelector('.profile__img').src = 'toddmotto.jpg';
3. template.querySelector('.profile__name').textContent = 'Todd Motto';
4. template.querySelector('.profile__social').textContent = 'Follow me on Twit-
   ter';
5. document.body.appendChild(template);
```

Koodiesimerkki 2. Template-elementin aktivointi JavaScriptillä [34].

<template>-elementtien käyttö tuo mukanaan muutamia hyödyllisiä ominaisuuksia. Se mahdollistaa muun muassa sen, että elementin sisältö ei reagoi tai näy selaimessa ennen kuin se aktivoidaan JavaScriptin toimesta. Sisältö, kuten komennot ja esitettävät kuvat eivät lataudu ennen kuin kyseistä sivupohjaa käytetään. Tämä johtaa siihen, että sivun latautuessa alustavia palvelinkutsuja voidaan suorittaa vähemmän. Hyödyllistä on myös se, että sivupohja voidaan sijoittaa mihin tahansa <head>-, <body>- ja <frameset>-elementtien sisälle. [33.]

Räätälöidyt elementit (Custom Elements)

Räätälöidyt elementit nimensä mukaisesti antavat kehittäjille mahdollisuuden luoda omia HTML-elementtejä ja muokata olemassa olevia elementtejä. Räätälöidyt elementit antavat standardoidun tavan luoda komponentteja, jotka sisältävät ominaiset tyylit ja toiminnot, JavaScriptin, HTML:n ja CSS:n avulla. [35.] Koodiesimerkissä 3 on esitetty räätälöidyn elementin luominen.

```
1. <template id="profileTemplate">
2.   <div class="profile">
3.     <img src="" class="profile__img">
4.     <div class="profile__name"></div>
5.     <div class="profile__social"></div>
6.   </div>
7. </template>
8. <script>
9.   var MyElementProto = Object.create(HTMLElement.prototype);
10.  window.MyElement = document.registerElement('user-profile', {
11.    prototype: MyElementProto
12.    // other props
13.  });
14.</script>
```

Koodiesimerkki 3. Räätälöidyn elementin luominen [34].

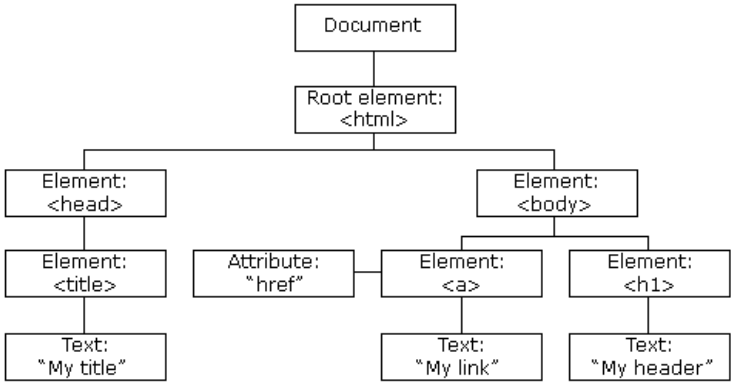
Esimerkin alussa näkyy edellisen esimerkin <template>-elementti. Rivillä 9 Object.create()-funktio palauttaa prototyypin, joka on laajennettu HTMLElementistä. HTMLElementistä laajentaminen varmistaa, että luotu elementti perii koko DOMin ohjelmointirajapinnan, ja kaikki ominaisuudet ja metodit, joita luodaan, lisätään elementin DOM-käyttöliittymään. Rivillä 10 document.registerElement()-funktio rekisteröi räätälöidyn elementin, jonka nimi on ’user-profile’, ja funktion sisällä määritetään elementin ominaisuudet. Tätä elementtiä käytetään sijoittamalla <user-profile>- ja </user-profile>-merkit haluttuun kohtaan. Räätälöityjen elementtien luomiseen liittyy muutamia sääntöjä. Elementin nimen tulee sisältää (-)-merkki, jotta HTML-jäsennin tunnistaa räätälöidyt elementit tavallisten elementtien seasta. Räätälöityjen elementtien käyttäminen edellyttää myös, että koodi sisältää aina elementin sulkumerkin (</user-profile>). [34; 35.]

Räätälöityjen elementtien liittäminen sivupohjiin ja varjo-DOMiin, josta kerrotaan seuraavana, luo perustan webkomponenteille. Niiden avulla uudelleen käytettävien komponenttien luominen ilman erillisiä kirjastoja on helppoa. [35.]

Varjo-DOM (ShadowDOM)

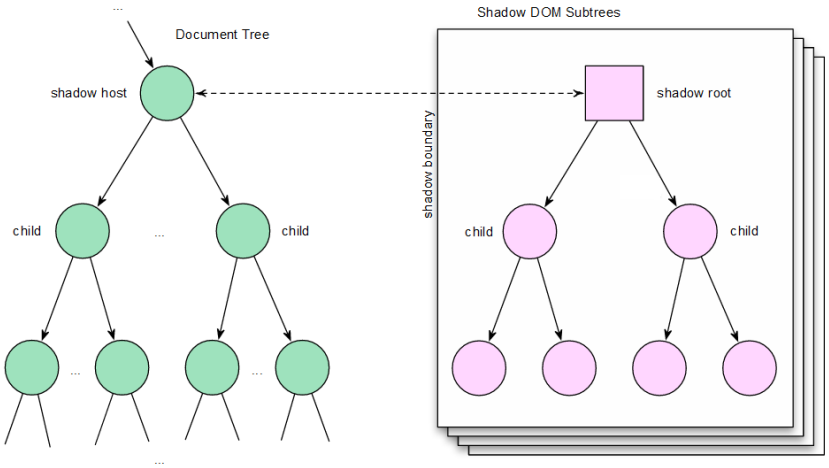
Kolmas webkomponenttien kehitteillä oleva standardi on niin kutsuttu varjo-DOM. Se mahdollistaa muun muassa elementin tyylitietojen sisällyttämisen vain kyseiseen elementtiin. Sivun latautuessa selain luo DOMin (Document Object Model) sivusta. DOM

kuvaa sivun rakenteen puuna, joka sisältää kaikki objektit, niin kutsutut solmut. [27.] Kuvassa 5 on esitetty esimerkki DOMista.



Kuva 5. HTML DOM (dokumenttioliomalli) [36].

Dokumenttipuu esittää tavallisen DOMin, jonka juurielementtinä toimii 'Document'. Tämä puu on käyttäjälle näkyvissä. Varjo-DOM on sisäinen alipuu, jota käyttäjä ei näe. Mikä tahansa solmu dokumenttipuussa voi olla "isäntänä" yhdelle tai usealle varjo-DOMille. [27.] Kuvassa 6 on esitetty dokumenttipuusta luotu varjo-DOM.



Kuva 6. Dokumenttipuu ja siitä luotu varjo-DOM [27].

Solmua, josta varjo-DOM luodaan, kutsutaan varjo-isännäksi (shadow host) Varjo-DOM sisältää varjo-juuren (shadow root), joka ei kuitenkaan ole varjo-isäntä-elementin lapsielementti. Varjo-juurella ei ole isäntää ollenkaan. Tämän vuoksi mikään varjo-DOMin sisällä oleva tieto ei vaikuta muihin puurakenteisiin. Varjo-DOM-alipuun täytyy aina olla linkitettynä olemassa olevaan elementtiin, varjo-isäntään. Tämä elementti voi olla HTML-

elementti, JavaScriptillä luotu elementti tai räätälöity elementti. [27; 37.] Koodiesimerkissä 4 on esitetty varjo-DOMin luominen <button>-elementille.

```
1. <button>Hello!</button>
2. <script>
3.   var host = document.querySelector('button');
4.   var root = host.createShadowRoot();
5. </script>
```

Koodiesimerkki 4. Varjo-DOMin luominen elementille [27].

Esimerkissä 4 rivillä 3 luodaan uusi muuttuja, joka saa arvokseen <button>-elementin. Tämän jälkeen rivillä 4 elementille luodaan varjo-DOM createShadowRoot()-funktiolla. Varjo-DOMin luomisen jälkeen varsinaisesta DOMista ja varjo-DOMista rakennetaan uusi dokumentti, jonka sisältö näytetään käyttäjälle.

Varjo-DOMeja käyttämällä elementtien sisältämät tyylit ja merkintäkieli voidaan kapseloida vain kyseistä elementtiä koskemaan, jolloin esimerkiksi tyylit eivät vuoda toisiin elementteihin. Voidaan ajatella, että räätälöidyt elementit ovat tapa luoda uusia HTML-elementtejä ja varjo-DOM on tapa tuottaa näiden elementtien merkintäkieli ja tyylit. [37.]

Tuonti-HTML (HTML Import)

Viimeinen webkomponenttien teknologia on tuonti-HTML. Tämän ominaisuuden avulla voidaan HTML-dokumenttiin linkittää toinen HTML-dokumentti. Sisällön liittämiseen HTML-dokumenttiin on olemassa useita tapoja. JavaScript liitetään <script>-merkinnän, CSS <link>-merkinnän, ja kuvat <img>-merkinnän avulla. Kuitenkaan ennen tuonti-HTML:ää ei ole ollut käytännöllistä keinoa liittää HTML-dokumentti toiseen HTML-dokumenttiin. [27.] Koodiesimerkissä 5 on esitetty esimerkki tuonti-HTML:n käytöstä.

```
<link rel="import" href="user-profile.html">

<!--
  <user-profile> now available
-->
```

Koodiesimerkki 5. Tuonti-HTML:n käyttö [34].

Tuonti-HTML käyttää <link>-elementtiä viittaamaan liitettävään tiedostoon.

Jokaista webkomponenttiteknologiaa voi käyttää erikseen tai sitten niitä voi yhdistellä keskenään. Näillä tekniikoilla pystytään ratkaisemaan monia ongelmia webkehityksen maailmassa. Kuitenkin jokaisessa webkomponentissa on vielä paljon kehittämisen varaa. [27.]

Webkomponentit tänään

Vaikka webkomponentit ovat olleet kehitteillä jo useita vuosia, kaikki neljä standardia ovat vielä kehitysvaiheissa. Nykypäivän kaikki selaimet eivät vielä tue kaikkia webkomponentteja. Kuvassa 7 on esitetty eniten käytettyjen selainten tuen tilanne kaikille webkomponenteille.

	Specced	Implementation				
		Polyfill	Chrome / Opera	Firefox	Safari	Edge
Templates			Stable	Stable	8	13
HTML Imports			Stable	On Hold	No Active Development	Vote
Custom Elements			Stable	Flag	10.1	Vote
Shadow DOM			Stable	Flag	10	Vote

Kuva 7. Webkomponenttien selaintukien tilanne 6.2.2017 [38].

Googlen Chrome-selain oli kaikista selaimista ensimmäinen, joka tukee kaikkien webkomponenttien käyttöä. Myös Opera-selain tukee kaikkia komponentteja. Muiden selainten tuki on kuitenkin vielä kehitysvaiheessa. Microsoftin Edge-selain ei tue muita komponentteja kuin sivupohjia. Microsoft on kuitenkin sanonut, että Edgen uuden DOMin kehittämisen jälkeen keskitytään räätälöityjen elementtien ja varjo-DOMin tuen lisäämiseen selaimeen. Mozilla on lisännyt kaikkien webkomponenttien tuet Firefox-selaimeen, mutta tuonti-HTML ei kuitenkaan ole ainakaan vielä käytettävissä. Applen Safari tukee kaikkia muita paitsi tuonti-HTML:ää. Koska osa webkomponenttien määritelmistä on vielä kehitysvaiheessa, niiden tukea ei haluta lisätä kaikkiin selaimiin, ainakaan vielä. Esimerkiksi sivupohjien ongelmana on, että vaikka sivupohja olisi reagoimaton ennen

aktivointia, jotkin JavaScript-tiedostot ja kuvat saattavat latautua myös ilman niiden aktivointia. Ongelmallista räätälöidyissä elementeissä on, että vielä ei ole tarkkaa määritelmää siitä, miten elementit tulisi määrittää räätälöidyiksi elementeiksi ja miten elementtejä tulisi laajentaa. Varjo-DOMin kohdalla ei ole tarkkaa määritelmää siitä, milloin sisältö tulisi liittää varjo-isännältä varjo-DOMiin. Osa selainten kehittäjistä uskoo, että tuonti-HTML ei tuo mukanaan mitään uutta, mitä ei voisi jotenkin toisin toteuttaa, ja siksi ne eivät ole lisänneet sen tukea selaimiin. Näistä ongelmista huolimatta teknologiaa kehitetään koko ajan, ja voidaan olettaa, että tämänkaltaiset ongelmat pystytään poistamaan webkomponenteista. [27.]

Selaintuen puute on isoin ongelma webkomponenttien ympärillä. Ennen kuin kaikkiin selaimiin on saatu kaikkien webkomponenttien tuki, ratkaisuna tähän ongelmaan on luotu kirjastoja, jotka sisältävät liitännäisiä (polyfills), joiden avulla selaimiin saadaan ominaisuudet webkomponenttien tukemiseksi. Suosituin liitännäispaketti on webcomponents.js (https://github.com/webcomponents/webcomponentsjs), joka antaa tuen webkomponenteille ja mahdollistaa niiden käytön moderneissa selaimissa. Tämä liitännäispaketti sisältää kaikkien webkomponenttien käyttöön tarvittavat osat, ja isona etuna on se, että liitännäiset aktivoituvat vain silloin, jos komponentin tuki puuttuu selaimesta. webcomponents.js-pakettia käyttää muun muassa kaksi suosituinta webkomponenttikirjastoa: Polymer ja X-tag. Näiden kirjastojen tarkoitus on auttaa kehittäjiä luomaan ja käyttämään webkomponentteja. [27.] Seuraavassa luvussa esitellään näitä kirjastoja ja niiden ominaisuuksia.

3.3 Webkomponenttikirjastot X-Tag & Polymer

Jotta webkomponenttien tuki voidaan lisätä kaikkiin selaimiin, tarvitaan joko liitännäinen tai webcomponents.js-liitännäispaketti tai komponenttikirjasto, joka sisältää tarvittavat liitännäiset. Webkomponenttien kehityksen helpottamiseksi on luotu jo muutamia kirjastoja. Näiden kirjastojen tarkoituksena on helpottaa kehittäjien työskentelyä webkomponenttien parissa. Webkomponenttien luominen ilman kirjastoja voi olla haastavaa, ja se vaatii paljon aikaa kehittäjiltä. Komponenttien luominen ilman kirjastoja on mahdollista, mutta jos tarkoituksena on luoda paljon erillisiä komponentteja, kirjastojen käyttö voi tehdä koodista yksinkertaisempaa ja luettavampaa ja työnkulusta sujuvampaa. Tässä luvussa esitetään kaksi tunnettua webkomponenttien kanssa työskentelyyn tarkoitettua kirjastoa, Microsoftin X-Tag (https://x-tag.github.io/) ja Googlen Polymer

(https://www.polymer-project.org/1.0/). X-Tagin ja Polymerin lisäksi on kehitetty muitakin kirjastoja, jotka eivät kuitenkaan ole vielä ainakaan pystyneet nousemaan X-Tagin ja Polymerin rinnalle. Yksi näistä on Bosonic, joka eroaa edellä mainituista kirjastoista siten, että sillä luodut komponentit muunnetaan JavaScriptiksi ja CSS:ksi. Muunnetut komponentit voidaan tämän jälkeen lisätä projektiin. Toinen kirjasto on Skate.js, joka mahdollistaa webkomponenttien kehittämisen minimalistisella tyylillä. [27; 39.] Tässä raportissa keskitytään kahteen suosituimpaan webkomponenttikirjastoon: Polymeriin ja X-Tagiin.

X-Tag

X-Tag on Daniel Buchnerin kehittämä pieni JavaScript-kirjasto, joka tuo selaimeen räätälöityjen elementtien tuen. Kun X-Tagin kehittäjä siirtyi Mozillalta työskentelemään Microsoftille, myös X-Tagin kehitys siirtyi Microsoftille. X-Tag on keskittynyt käyttämään CustomElement-ohjelmointirajapintaa kaikkien webkomponenttien rajapintojen sijaan. X-Tag on siis riippuvainen ainoastaan räätälöityjen elementtien teknologiasta. Tämä on mahdollista, koska kaikkia webkomponenttiteknologioita voi käyttää myös toisista erillään. [27.]

Tärkein X-Tagin metodeista on xtag.register(), jolla uusi elementti rekisteröidään. Rekisteröinnin yhteydessä määritellään elementille tyypilliset ominaisuudet ja toiminnallisuudet. X-Tag käyttää elementtien rekisteröimiseen document.registerElement()-funktiota, joka on sisällytetty xtag.register()-funktioon. [40.] Koodiesimerkissä 6 on nähtävillä esimerkki räätälöidyn elementin luomisesta X-Tagilla.

```
xtag.register('my-element', {
  content: '<p>Hello there</p>'
});
```

Koodiesimerkki 6. Elementin luominen X-Tagilla [40].

Vaikka X-Tag on keskittynyt räätälöityjen elementtien teknologiaan, myös sivupohjien ja varjo-DOMin käyttäminen onnistuu X-Tagin kanssa. Jos X-Tagin kanssa halutaan käyttää sivupohjaa ja/tai varjo-DOMia, se onnistuu. [40.] Koodiesimerkissä 7 on esitetty sivupohjien ja varjo-DOMin käyttö X-Tagin kanssa.

```
1. <template id="my-template">
2.   <p>Hello there</p>
3. </template>
4. xtag.register('my-element', {
5.   shadow: '<p>Hello There</p>'
6. });z
```

Koodiesimerkki 7. Sivupohjan ja varjo-DOMin käyttö X-Tagilla [40].

Sivupohjan luomiseen vaaditaan vain sivupohjan määrittäminen ennen elementin rekisteröintiä, ja sen jälkeen sen sisältö kloonataan elementin DOMiin (koodiesimerkki 8).

```
1. <template id="my-template">
2.   <p>Hello there</p>
3. </template>
4. var tpl = document.getElementById('my-template').content;
5. xtag.register('my-element', {
6.   lifecycle: {
7.     created: function () {
8.       this.appendChild(tpl.cloneNode(true));
9.     }
10.  }
11.});
```

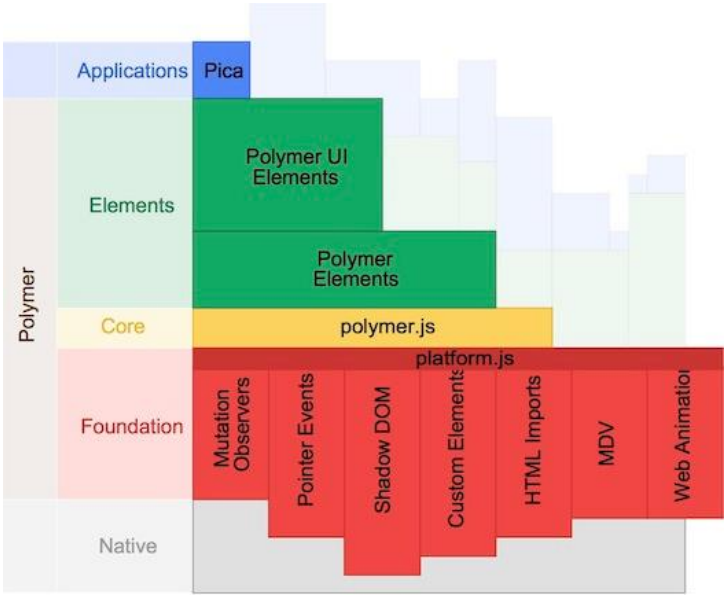
Koodiesimerkki 8. Sivupohjan määrittäminen ja sisällön luominen X-Tagilla [40].

Esimerkissä määritetään ensin sivupohja, jonka id:ksi asetetaan ”my-template”. Tämän jälkeen elementin rekisteröinnin yhteydessä sivupohjan sisältö määritetään.

Jos X-Tagin kanssa halutaan kapseloida elementin tyylit käyttämällä varjo-DOMia, se onnistuu yksinkertaisesti ’shadow’-ominaisuuden lisäämisellä elementin määrittelyyn, niin kuin koodiesimerkissä 7 rivillä 5 on tehty. X-Tagin vahvuutena voidaan pitää sitä, että sen keskittyessä räätälöityihin elementteihin koodi pysyy pienenä ja sen suorittamiseen kuluu vähemmän aikaa ja silti tarvittaessa sivupohjien ja varjo-DOMin käyttäminen onnistuu. Omien komponenttien luomisen lisäksi kehittäjät voivat käyttää Mozillan Brick-kirjaston valmiita räätälöityjä elementtejä. [27; 40.]

Polymer

Polymer on Googlen kehittämä kirjasto webkomponenttien luomiseen. X-Tagista poiketen Polymer hyödyntää kaikkien webkomponenttien teknologioita, ja sen tarkoituksena on ohjata kehittäjää käyttämään kaikkia teknologioita yhdessä. Polymer toimii kevyenä kerroksena webkomponenttien päällä yhdistäen kaikkien teknologioiden ohjelmointirajapinnat toisiinsa. [41.] Polymer koostuu neljästä kerroksesta, jotka on havainnollistettu kuvassa 15.



Kuva 8. Polymerin rakenne kerroksittain esitettynä [41].

Ensimmäinen kerros (Native) sisältää ominaisuudet, jotka ovat natiivisti kaikissa moderneissa selaimissa. Toinen kerros (Foundation) luo pohjan Polymerille: se sisältää liitännäiset, joiden avulla webkomponenttien tuki saadaan lisättyä selaimiin, joissa ne eivät ole natiivisti tuettuina. Kolmas kerros (Core) on varsinainen polymer.js-kirjasto, jossa muun muassa määritellään elementtien perusrakenne, jotta ne hyödyntäisivät kahden alemman kerroksen ominaisuuksia. Viimeinen kerros (Elements) sisältää valmiiksi rakennetut elementit ja käyttöliittymäkomponentit, joita kehittäjät voivat käyttää projekteissaan. [41.]

Elementin rakentaminen ja rekisteröiminen Polymerilla on kuvattu koodiesimerkissä 9.

```
1. <dom-module id="element-name">
2.   <template>
3.     <style>
4.       /* CSS rules for your element */
5.     </style>
6.     <!-- local DOM for your element -->
7.     <div>{{greeting}}</div> <!-- data bindings in local DOM -->
8.   </template>
9.   <script>
10.    // element registration
11.    Polymer({
12.      is: "element-name",
13.      // add properties and methods on the element's prototype
14.      properties: {
15.        // declare properties for the element's public API
16.        greeting: {
17.          type: String,
18.          value: "Hello!"
19.        }
20.      }
21.    });
22.  </script>
23. </dom-module>
```

Koodiesimerkki 9. Uuden elementin rekisteröinti ja luominen Polymerilla [42].

Jotta uudelle elementille voidaan luoda paikallinen DOM, pitää elementti rakentaa <dom-module>-elementin sisälle. Niin kuin räätälöityjen elementtien määritelmässä on määritelty, elementit rekisteröidään document.registerElement()-funktiolla. Myös Polymer käyttää tätä funktiota elementtien rekisteröimiseen, ja se on sisällytetty Polymer()-funktioon, joka alkaa esimerkissä rivillä 11. Elementin tyylit määritellään <style>-elementin sisällä, ja sivupohja luodaan <template>-elementin sisälle. Elementin toiminnallisuudet määritellään <script>-elementin sisällä. Elementin luomisessa on välttämätöntä määritellä elementin nimi Polymer()-funktion sisällä, jotta se rekisteröityisi oikein. Saman funktion sisällä voidaan määritellä myös elementin toiminnot ja toimintojen elinkaari (lifecycle). Rekisteröinnin yhteydessä elementille voidaan määritellä ominaisuuksia, jotka tukevat muun muassa kaksisuuntaista datakytkentää eri elementtien välillä. Polymer()-

funktiota käyttämällä elementti rekisteröidään selaimen käyttöön, jolloin se palauttaa muodostimen (constructor), jonka avulla voidaan luoda uusia elementin ilmentymiä (instances). [43; 44.]

Koska Polymer on keskittynyt kaikkien webkomponenttien teknologioihin, sen täytyy sisältää kaikki tarvittavat liitännäiset, jotta komponenttien tuki saadaan kaikkiin selaimiin. Kokonaisuudessaan Polymer-kirjaston koko on suhteellisen suuri: 163 kilotavua. Isoin yksittäinen osa kirjastossa on Varjo-DOM-liitännäinen, ja sen iso koko myös hidastaa Polymerin latausnopeutta. Ratkaisuna Polymerin kehittäjät ovat kehittäneet niin sanotun varjoisan DOMin (ShadyDOM), joka on varjo-DOMin kilpailija. Varjoisa DOM on varjo-DOMia pienempi, mutta siinä ei ole kaikkia varjo-DOMin ominaisuuksia. Varjo-DOMin tarkoituksena on piilottaa sisäiset alipuut käyttäjältä. Varjoisa DOM ei tätä kuitenkaan tee, mutta se aiheuttaa tarpeeksi samankaltaisia tuloksia kuin varjo-DOM, jotta Polymeria voi käyttää ilman raskasta varjo-DOM-liittännäistä. Tämän seurauksena myös Polymer-kirjaston koko pienenee huomattavasti. [27; 45.]

Polymerin suurena etuna on, että se sisältää kaikkien webkomponenttien tukemiseen tarvittavat liitännäiset ja että sen käyttäminen muiden kirjastojen kanssa on saumatonta. Polymer-kirjaston dokumentointi on toteutettu hyvin, ja se sisältää paljon esimerkkejä eri ominaisuuksista. Lisäksi Polymerin kehittäjät ja käyttäjät ovat kehittäneet lukuisia räätälöityjä elementtejä, jotka on koottu webcomponents.org-sivustolle (https://www.webcomponents.org/), ja ne ovat vapaasti muiden kehittäjien käytettävissä. [41; 42.]

Polymer ja X-Tag

Polymer- ja X-Tag-webkomponenttikirjastot eroavat toisistaan monin osin. X-Tag on keskittynyt vain yhteen webkomponenttien teknologiaan, kun taas Polymer kattaa kaikkien komponenttien teknologiat. Teknisten ominaisuuksien lisäksi kirjastot eroavat muun muassa niitä ympäröivien yhteisöjen koossa ja dokumentoinnin laajuudessa. Polymer on muunkin kuin kokonsa puolesta suurempi kuin X-Tag. Yhteisö Polymerin ympärillä on paljon suurempi kuin X-Tagilla, ja yksi suurimmista vaikuttajista kirjastoja ja ohjelmistokehyksiä valittaessa onkin juuri yhteisö ja sen koko. Suuri yhteisö tarkoittaa usein enemmän kysyttyjä ja vastattuja kysymyksiä, enemmän erilaisia ohjeistuksia ja enemmän ratkaistuja ongelmia. Yhteisön koosta voi saada osviittaa esimerkiksi tarkasteltaessa kunkin kirjaston ja ohjelmistokehyksen GitHub-tilastoja. Myös esimerkiksi YouTube-hakutu-

lokset ja kysytyt kysymykset Stack Overflow -sivustolla antavat hyvän käsityksen yhteisön koosta ja kyseisen kirjaston tai ohjelmistokehyksen suosiosta. Taulukkoon 1 on koottu Polymerin ja X-Tagin GitHub-tilastot ja kysyttyjen kysymysten määrä Stack Overflow'ssa.

Taulukko 1. Polymerin ja X-Tagin GitHub-tilastot ja kysymykset Stack Overflow'ssa. Muokattu GitHubista ja Stack Overflow'sta [46; 47; 48; 49].

Mittari	Polymer.js, kpl	X-Tag, kpl
GitHub-tähdet (stars)	17 116	366
GitHub-avustajat (contributors)	107	8
Avoimet ongelmat GitHubissa	606	0
Ratkaistut ongelmat GitHubissa	2 619	42
Kysymykset Stack Overflow'ssa	6 461	28

Kaikista taulukon kohdista voidaan päätellä, että Polymerin yhteisö on suurempi kuin X-Tagilla. Polymerilla on GitHubissa enemmän osallistuvia kehittäjiä, ja se on saanut paljon enemmän tähtiä kuin X-Tag. Avustajien, avoimien ongelmien ja kysyttyjen kysymysten määrä kertoo siitä, että ihmiset ovat kiinnostuneet käyttämään Polymeria ja uudet ongelmat pyritään ratkaisemaan yhteisön voimin.

Yhteisön laajuuden lisäksi sopivimman kirjaston ja ohjelmistokehyksen valintaan vaikuttaa muun muassa dokumentointi ja tiedostojen koko. Polymer on ymmärrettävästikin suurempi kuin X-Tag tiedostojen kokoa katsottaessa, koska se sisältää kaikkien webkomponenttien liitännäiset. X-Tagin kooksi minimoinnin ja pakkauksen jälkeen jää 20 kilotavua ja Polymerin kooksi varjo-DOMin kanssa 44 kilotavua. Kuitenkin, jos varjo-DOM ei ole välttämätön ja varjoisa DOM riittää, Polymerinkin koko pienenee. [27; 50.]

Tähän insinöörityöhön sopivaa webkomponenttikirjastoa valittaessa iso vaikutus oli myös kirjastojen dokumentoinnilla ja niiden laajuudella. Polymerin ja X-Tagin dokumentointia vertailtaessa ensimmäisellä on huomattavasti kattavampi dokumentointi. Polymerin sivustolla osoitteessa https://www.polymer-project.org/ on hyvät esimerkit elementtien ja verkkosovellusten rakentamiseen, ja aloitustutoriaalien avulla aloittelijakin pääsee helposti kiinni webkomponenttien kehitykseen Polymerilla. Sivusto kertoo kattavasti kaikista Polymerin ominaisuuksista, ja siellä on havainnollistettu koodiesimerkein lähes

kaikkien tekniikoiden käyttö. X-Tagin dokumentointi osoitteessa http://x-tag.github.io/overview on huomattavasti suppeampi kuin Polymerilla. Sivustolla on annettu vain muutama koodiesimerkki X-Tagin käytöstä, eikä sen ominaisuuksista ole kerrottu yhtä kattavasti kuin kilpailijansa.

Insinöörityön kannalta oleellista oli valita sopiva webkomponenttikirjasto työtä varten. Tässä vaiheessa päätös kirjaston valinnasta oli jo aika selkeä, mutta ennen lopullista valintaa toteutettiin pienet esimerkit molempien kirjastojen avulla. Tarkoituksena oli luoda räätälöidyt elementit, joiden avulla sivulle lisätään Google Maps -kartat. Esimerkeissä käytettiin Google Maps Embed -ohjelmointirajapintaa, joka mahdollistaa interaktiivisen kartan lisäämisen sivustolle yksinkertaisen http-kyselyn avulla. Seuraavaksi esitellään nämä esimerkit, jotka luotiin Polymerilla ja X-Tagilla.

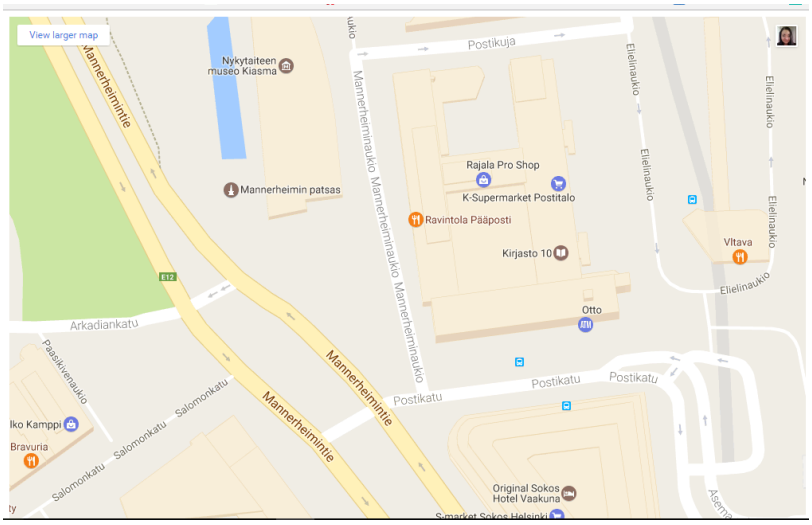
X-Tag

X-Tag-kirjaston saa helposti ladattua osoitteesta http://x-tag.github.io/builds. Ladattavissa olevia versioita kirjastosta on kaksi: liitännäisen kanssa tai ilman. Esimerkkiä varten ladattiin versio liitännäisten kanssa, jotta omia liitännäisiä ei tarvitsisi luoda. Koodiesimerkissä 10 on X-Tagilla toteutettu räätälöity elementti.

```
1.  <!DOCTYPE html>
2.  <html>
3.    <head>
4.      <script src="x-tag-core.min.js"></script>
5.    </head>
6.    <body>
7.      <google-map latitude longitude></google-map>
8.      <script>
9.        var API_KEY = 'XXXXXXXXXX'; //API key
10.       xtag.register('google-map', {
11.         lifecycle: {
12.           created: function() {
13.             var iframe = document.createElement('iframe');
14.             iframe.width = 1000
15.             iframe.height = 1000;
16.             iframe.frameBorder = 0;
17.             iframe.src = 'https://www.google.com/maps/embed/v1/view?key=' + API_KEY + '&cen
    ter=' + this.latitude + ',' + this.longitude + '&maptype=roadmap&zoom=18';
18.             this.appendChild(iframe);
19.           }
20.         },
21.         accessors: {
22.           latitude: {
23.             attribute: {},
24.             set: function(value) {
25.               this.xtag.data.latitude = value;
26.             },
27.             get: function() {
28.               return this.getAttribute('latitude') || "60.170833";
29.             }
30.           },
31.           longitude: {
32.             attribute: {},
33.             set: function(value) {
34.               this.xtag.data.longitude = value;
35.             },
36.             get: function() {
37.               return this.getAttribute('longitude') || "24.9375";
38.             }
39.           }
40.         }
41.       });
42.     </script>
43.   </body>
44. </html>
```

Koodiesimerkki 10. X-Tagilla toteutettu räätälöity elementti. Muokattu Codepen-esimerkistä [51].

Koodiesimerkissä rivillä 4 sovellukseen sisällytetään ladattu X-Tag-kirjasto <script>-elementin avulla. Rivillä 9 alkaa varsinaisen sovelluksen rakentaminen. Jotta Google Maps Embed -ohjelmointirajapintaa voi käyttää, tarvitaan kehittäjälle räätälöity avain, jonka hankkimiseen on ohjeet ohjelmointirajapinnan dokumentaatiossa (https://developers.google.com/maps/documentation/embed/guide#api_key). Räätälöidyn elementin rekisteröinti alkaa rivillä 10: siinä määritellään kehitettävän elementin nimi, tässä esimerkissä ’google-map’. Lifecycle()-funktion sisälle määritellään elementin elinkaari, ja created()-funktio ja sen sisällä olevat ominaisuudet toteutuvat, kun ’google-map’-elementti on luotu. Esimerkissä ’google-map’-elementin luomisen jälkeen created()-funktion sisällä luodaan <iframe>-elementti, jonka koko ja sisältö määritellään samalla. <iframe>-elementin käyttöä suositellaan Embed-ohjelmointirajapinnan kanssa, jotta kartta saadaan helposti istutettua sivulle [52]. <iframe>-elementin sisällön lähde määritellään rivillä 17, jossa toteutetaan http-kysely Googlelta. Kyselyssä määritellään käytettävä ohjelmointirajapinta ja sen versio, API-avain, karttatyyppi, zoomaus ja esitettävän kartan keskikohta. Tämän jälkeen <iframe>-elementti lisätään sivustolle appendChild()-funktiolla. Rivillä 21 oleva ’accessors’-objekti mahdollistaa muun muassa räätälöidyn elementin ominaisuuksien (attribute) hallinnoinnin. Esimerkissä ’latitude’- ja ’longitude’-ominaisuudet asetetaan ’accessors’-objektin sisällä. get()-funktioissa määritellään arvot, ja set()-funktioiden avulla arvot asetetaan vastaamaan elementin attribuutteja. ’latitude’- ja ’longitude’-arvoja käytetään esimerkissä asettamaan kartan keskipiste Helsingin keskustaan. Arvot saadaan lisättyä kyselyyn helposti lisäämällä ’this.latitude’- ja ’this.longitude’-muuttujat, niin kuin rivillä 17 on tehty. [53.] Kuvassa 9 on esitetty esimerkin lopputulos kuvakaappauksena.



Kuva 9. X-Tagilla toteutettu Google Maps -karttaelementti.

Polymer

Polymer saadaan ladattua GitHubista osoitteesta https://github.com/polymer/polymer/releases. Ladattu kansio sisältää standardiversion lisäksi Polymerin mini- ja mikroversiot, joiden sisältämät ominaisuudet on karsittu minimiin. Tämän lisäksi täytyy ladata webcomponents.js-liitännäinen käyttämällä Boweria (https://bower.io/), joka on ohjelmoinnissa käytetty ohjelma kirjastojen ja liitännäisten lataamiseen ja hallinnoimiseen [54]. webcomponents.js-liitännäinen ladataan käyttämällä komentorivillä komentoa

```
bower install --save webcomponents/webcomponentsjs
```

Esimerkissä käytetään Polymerin standardiversiota. Koodiesimerkeissä 11 ja 12 on esitettynä räätälöidyn karttaelementin luominen Polymerilla.

```
1.  <!DOCTYPE html>
2.  <html>
3.    <head>
4.      <link rel="import" href="polymer.html">
5.      <script                  src="bower_components/webcomponentsjs/webcompo-
    nents.js"></script>
6.      <link href="style.css" rel="stylesheet">
7.    </head>
8.    <body>
9.      <dom-module id="google-map">
10.       <template>
11.         <style>
12.           iframe{
13.             width: 1000px;
14.             height: 1000px;
15.           }
16.         </style>
17.         <iframe src="{{src}}"></iframe>
18.       </template>
```

Koodiesimerkki 11. Räätälöidyn elementin luominen (osa 1/2).

<head>-elementin sisällä Polymer-kirjasto ja webcomponents.js-liitännäinen sisällytetään sivun käyttöön. Rivillä 9 alkaa räätälöidyn elementin rakentaminen <dom-module>-elementillä. <dom-module>:n sisällä määritellään räätälöidyn elementin id, ’google-map’,

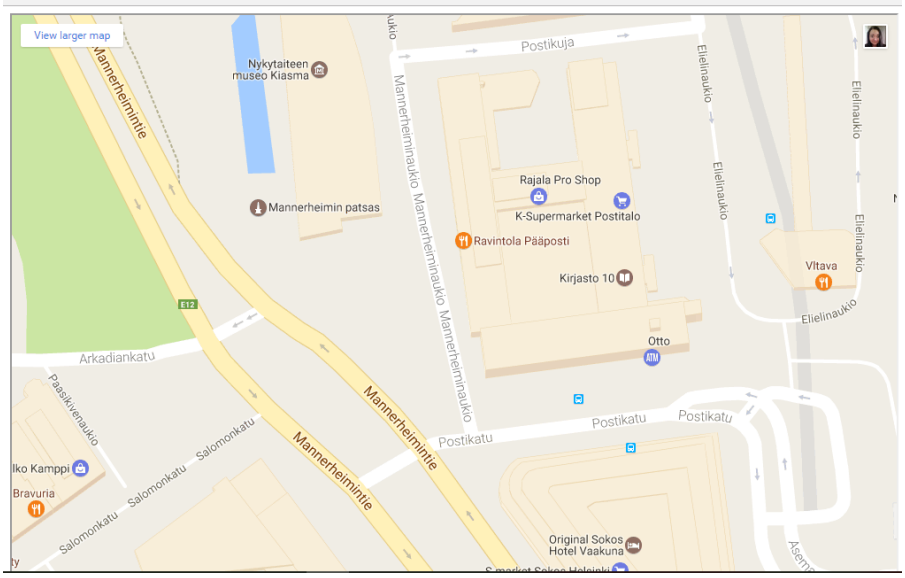
ja lisätään elementille sivupohja ja tyylit. Sivupohjan sisällä määritetään <iframe>-elementti ja sille ominaiset tyylit. <iframe>:n lähde määritellään JavaScriptillä, ja se sidotaan ’src’-attribuuttiin käyttämällä kaksoisaaltosulkuja.

```
19.       <script>
20.         Polymer({
21.           is: 'google-map',
22.           properties: {
23.             latitude: {
24.               type: Number,
25.               value: '60.170833'
26.             },
27.             longitude: {
28.               type: Number,
29.               value: '24.9375'
30.             },
31.             src: {
32.               type: String
33.             }
34.           },
35.           ready: function(){
36.             this.src          =          'https://www.google.com/maps/em-
    bed/v1/view?key=XXXXXXXX&center=' + this.latitude + ',' + this.longitude +
    '&maptype=roadmap&zoom=18'
37.           }
38.         });
39.       </script>
40.     </dom-module>
41.     <google-map></google-map>
42.   </body>
43. </html>
```

Koodiesimerkki 12. Räätälöidyn elementin luominen (osa 2/2).

Räätälöity elementti rekisteröidään Polymer()-funktiota käyttämällä. ’is’-ominaisuus määrää luotavan HTML-elementin nimen, ja sen on oltava sama kuin <dom-module>:n id [44]. Riviltä 22 lähtien elementille asetetaan halutut ominaisuudet. Ominaisuuksille voidaan asettaa muun muassa tyyppi ja arvo. Esimerkissä ’latitude’- ja ’longitude’-ominaisuuksille asetetaan tyypeiksi ’Number’ ja arvoiksi numeroarvot, ’src’-ominaisuudelle asetetaan tyypiksi ’String’. Ominaisuuksien asettamisen jälkeen rivillä 35 luodaan ready()-funktio, jota kutsutaan, kun luotavan elementin DOM on alustettu. Esimerkissä ready()-funktiossa lähetetään http-kysely Google Maps Embed -ohjelmointirajapinnalle. Kartan

keskipisteen koordinaatit asetetaan ’this.latitude’- ja ’this.longitude’-ominaisuuksien arvoilla. Kysely tallentuu ’this.src’-muuttujaan, joka sidotaan datakytkennällä aiemmin mainittuun ’{{src}}’-attribuuttiin. Näin vastauksena saatu kartta saadaan asetettua <iframe>-elementtiin. Lopuksi rivillä 41 on esitetty luodun elementin käyttö. Kuvassa 10 on esitetty esimerkin lopputulos.



Kuva 10. Polymerilla toteutettu karttaelementti.

Niin kuin esimerkeistä huomaa, myös syntaksien väliset erot ovat huomattavat. X-Tagissa elementin luominen toteutetaan kokonaan pelkällä JavaScriptillä, kun taas Polymerissa hyödynnetään JavaScriptin lisäksi HTML:ää. Polymerin syntaksi vaikuttaa minusta muutenkin selkeämmältä ja helpommin ymmärrettävältä. Kirjastojen ominaisuuksia, dokumentointia, yhteisön kokoja ja luotuja esimerkkejä vertailemalla varsinainen insinöörityö päädyttiin toteuttamaan käyttämällä Polymer-kirjastoa. Vaikka Polymer onkin kooltaan isompi ja raskaampi, se on monin osin järkevämpi vaihtoehto tähän työhön kuin X-Tag. Polymerin dokumentaatio on kattavampi, yhteisö isompi ja syntaksi helpompi oppia hyvien esimerkkien avulla. Seuraavissa luvuissa esitellään hieman enemmän Polymerin ominaisuuksia, toteutettu insinöörityö ja tehdyn työn tulokset.

4 Verkkosovelluksen kehittäminen Polymerilla

4.1 Polymerin ominaisuuksia

Polymer.js on suhteellisen uusi kirjasto, joka antaa kehittäjille kokoelman ominaisuuksia räätälöityjen elementtien kehittämiseen. Näiden ominaisuuksien tarkoituksena on nopeuttaa ja helpottaa uusien, vakio-DOM-elementtien (esimerkiksi - ja <button>) kaltaisten elementtien luomista. Polymerilla luoduilla elementeillä on samoja ominaisuuksia kuin vakio-DOM-elementeillä. Ne voivat olla

- luotu muodostimen (constructor) tai document.createElement()-funktion avulla
- määritelty attribuuttien ja ominaisuuksien (properties) avulla
- attribuuttien ja ominaisuuksien muutoksiin reagoivia
- paikallisesti tyyliteltyjä
- funktioihin reagoivia [42].

Polymerin kehittäjät ovat jakaneet kirjaston ominaisuudet eri kategorioihin, jotka on kuvattu taulukossa 2.

Taulukko 2. Polymer-kirjaston tarjoamat ominaisuudet kategorioittain [42].

Ominaisuus	Kuvaus
Rekisteröinti ja elementin elinkaari (Registration and lifecycle)	Elementti rekisteröidään, ja se tuottaa callback-funktioita, joilla hallitaan elementin elinkaarta.
Elementin ominaisuudet (Declared properties)	Voidaan konfiguroida attribuuttien avulla. Voivat reagoida muutoksiin, datakytkentään ja muiden attribuuttien muutoksiin.
Paikallinen DOM (Local DOM)	Elementin luoma ja hallitsema DOM.
Tapahtumat (Events)	Elementtiin liitettävät tapahtumat, esimerkiksi hiiren klikkaukset, joihin se reagoi.
Datakytkentä (Data binding)	Ominaisuuksien kytkentä esimerkiksi eri elementtien välillä. Myös attribuutteihin kytkeminen mahdollista.
Elementin käytös (Behaviors)	Uudelleen käytettäviä koodimoduuleja, joita voidaan liittää elementteihin.
Yleiset funktiot (Utility functions)	Yleisiin tehtäviin tarkoitetut yleiset funktiot.
Kokeelliset ominaisuudet ja elementit (Experimental features and elements)	Kokeelliset sivupohja- ja tyyliominaisuudet. Ominaisuuksien kerrostaminen mikro-, mini- ja standardiominaisuuksiin.

Polymer tarjoaa siis räätälöityjen elementtien luomisen yhteydessä kokoelman erilaisia ominaisuuksia elementtien luomiseen ja niiden hallitsemiseen. Kehittäjä voi määrittää räätälöidyille elementille elinkaaren, ominaisuudet, ja paikallisen DOMin. Räätälöity elementti voi reagoida muun muassa käyttäjän tekemiin pyyntöihin ja datakytkentään elementtien ja attribuuttien välillä. Polymer-kirjaston sivulla on koottu kaikista ominaisuuksista laajat kuvaukset, ja jokaisesta ominaisuudesta on sivulla esimerkkejä. Polymer-kirjasto tarjoaa myös joukon työkaluja, joilla Polymer-elementtien kehittäminen, rakentaminen ja optimointi ovat mahdollisia. Näistä työkaluista hyvä esimerkki on Polymer CLI (command-line interface), joka sisältää kehityskanavan (build pipeline), vakiokoodit (boilerplate) elementtien ja sovelluksien luomiseen ja palvelimen kehitykseen ja testityökalun elementtien testaamiseen [55]. Kaikki Polymerin työkalut ovat listattuna osoitteessa https://www.polymer-project.org/1.0/docs/tools/overview.

Helpottaakseen kehittäjän työtä entisestään Polymer-tiimi on kehittänyt kokoelman valmiita uudelleen käytettäviä elementtejä, joita verkkosovelluksen kehittäjä voi hyödyntää. Valmiit elementit on jaettu yhdeksään kategoriaan:

- App Elements – kokoelma elementtejä, jotka voivat olla hyödyllisiä sovelluksien rakentamisessa, esimerkiksi <app-route> reititykseen
- Iron Elements – elementit, joilla voidaan luoda sovellukseen perustoiminnallisuuksia, kuten Ajax-kyselytoiminto (<iron-ajax>)
- Paper Elements – kokoelma Googlen Material Design -käyttöliittymäelementtejä
- Google Web Components – elementit, joita hyödynnetään Googlen ohjelmointirajapintojen ja palveluiden kanssa
- Gold Elements – sähköiseen kaupankäyntiin tarkoitetut elementit, kuten luottokortin varmistukseen toteutettu tekstikenttä
- Data Elements – kokoelma elementtejä datan varastointiin ja käsittelyyn
- Layout Elements – kokoelma elementtejä sovelluksen ulkoasun toteuttamiseen, esimerkiksi elementti listan toteuttamiseen
- Platinum Elements – elementit, jotka tekevät verkkosivusta sovelluksen kaltaisen, esimerkiksi elementti bluetoothin käyttämiseen
- Molecules – elementit muiden kirjastojen paketointiin, jotta niitä on helpompi käyttää Polymerin kanssa. [56.]

Kaikki luodut elementin on koottu www.webcomponents.org-sivustolle. Sivusto sisältää Polymer-elementtien lisäksi myös yksittäisten kehittäjien tekemiä elementtejä. Esimerkiksi kun hakee Ajax-kutsuihin tarkoitettuja elementtejä, hakutuloksiin ilmestyy seitsemän kehitettyä elementtiä Polymerin <iron-ajax>-elementin lisäksi. Useista elementeistä on tehty dokumentaatio sen käytöstä, ja esimerkkidemot havainnollistavat hyvin elementin käyttöä omassa sovelluksessa. Tässä insinöörityössä hyödynnettiin sekä Polymer-tiimin kehittämiä elementtejä että muiden kehittäjien tekemiä elementtejä.

Polymerista on kaiken kaikkiaan julkaistu vuosien kuluessa 98 versiota, ja koko ajan kirjastoa kehitetään eteenpäin ja siihen tehdään parannuksia. Tällä hetkellä Polymerin uusin vakaa versio on 1.8.1, mutta myös ensimmäiset versiot Polymer 2.0:sta on juuri julkaistu. Tässä insinöörityössä käytettiin Polymerin versiota 1.7.1, joka oli uusin versio työtä aloitettaessa. [57.] Seuraavissa luvuissa esitellään kehitettyä verkkosovellusta, sen komponentteja ja toiminnallisuuksia.

4.2 Toteutetun sovelluksen komponentit ja toiminnallisuudet

Insinöörityön tarkoituksena oli toteuttaa yksinkertainen verkkosovellus hyödyntäen webkomponentteja, ja sen avulla tutkia, miten webkomponentit vaikuttavat verkkosovelluksen kehittämiseen. Tässä luvussa esitellään sovelluksen toiminnallisuuksia ja niiden toteuttamiseen käytettyjä elementtejä. Esimerkeissä esitellään Polymerin <iron-ajax>-, <dom-repeat>-, <app-route>-, <iron-pages>-, <google-map>- ja <geo-location>-elementit. Näitä kaikkia elementtejä käytettiin isomman sovelluksen toiminnallisuuksien toteuttamiseen, ja näiden elementtien lisäksi myös valmiita ulkoasuelementtejä hyödynnettiin. Kaikki esimerkeissä käytetyt koodit ovat osuuksia insinöörityönä toteutetusta sovelluksesta.

<iron-ajax>

Yksi tärkeimmistä asioista JavaScript-kirjastoissa on se, miten ne kommunikoivat palvelinpuolen kanssa. Usein kommunikointi on toteutettu REST API -tekniikalla. REST API:lla tarkoitetaan ohjelmointirajapintaa, joka käyttää http-pyyntöjä datan lähettämiseen, vastaanottamiseen ja poistamiseen [58]. Kommunikointi asiakaspuolen ja REST-ohjelmointirajapinnan välillä on helppo toteuttaa käyttäen Ajaxia, johon Polymer-tiimi on kehittänyt myös oman elementtinsä helpottaakseen sovelluskehittäjien työtä. Koodiesimerkissä 13 esitellään, miten <iron-ajax>-elementtiä käytetään pyyntöjen lähettämiseen Foursquaren REST-ohjelmointirajapinnalle. Foursquare on palvelu, josta käyttäjä voi etsiä itseään lähellä olevia paikkoja, ja ohjelmointirajapinta antaa kehittäjille pääsyn paikkatietoihin, joita voi hyödyntää omissa sovelluksissa [59].

```
1.  <link rel="import" href="../bower_components/polymer/polymer.html">
2.  <link rel="import" href="../bower_components/iron-ajax/iron-ajax.html">
3.  <iron-ajax
4.    auto
5.    url="https://api.foursquare.com/v2/venues/search"
6.    params="{{ajaxParams}}"
7.    last-response="{{data}}"
8.    handle-as="json"
9.    on-response="venuesLoaded">
10. </iron-ajax>

11. //Parametrien määritys Polymer()-funktion sisällä
12. ajaxParams: {
13.   type: String,
14.   computed: 'processParams(client_id, client_secret, v, limit, ll, catego-
    ryId, radius)'
15. }
```

Koodiesimerkki 13. Ajax-kutsun tekeminen <iron-ajax>-elementtiä käyttämällä.

Ennen <iron-ajax>-elementin käyttöä pitää Polymer-kirjasto ja elementti ladata. Tässä esimerkissä lataukset on suoritettu Bower-ohjelman avulla, jolloin tarvittavat osat latautuvat automaattisesti bower_components-kansioon. Polymer-kirjasto ja <iron-ajax>-elementti lisätään tuonti-HTML-tekniikalla riveillä 1 ja 2. Varsinaisen <iron-ajax>-elementin käyttö alkaa rivillä 3, jolloin elementin sisällä määritellään halutut attribuutit. Mahdollisia attribuutteja on lukuisia, ja kaikki vaihtoehdot on esitelty elementin dokumentaatiossa https://www.webcomponents.org/element/PolymerElements/iron-ajax/iron-ajax. ’auto’-attribuutti määrää sen, että silloin, jos hakuosoite tai joku parametreista puuttuu, suoritetaan automaattisesti uusi Ajax-kutsu. ’url’-attribuuttiin määritetään, mistä haluttu tieto haetaan. Tässä haetaan Foursquaren palvelusta paikkatietoja. ’params’-attribuutti saa arvokseen ’ajaxParams’-muuttujan, jonka arvo määritellään Polymer()-funktion sisällä. ’ajaxParams’ saa arvokseen funktion processParams(), jonka sisällä on halutut hakuparametrit, tässä tapauksessa tunnistukseen käytettävät ’client_id’ ja ’client_secret’, versio (v), ’limit’ määrittämään hakutulosten määrän, paikkatiedot (ll), haetun kategorian tunnus ja hakualue. ’last-response’-attribuutti määrittää, että saadut hakutulokset tallennetaan ’data’-muuttujaan, ja ’handle-as’ määrittää haetun tiedon formaatin. ’on-response’ kutsuu venuesLoaded()-funktiota, jossa hakutulokset tallennetaan taulukkoon. Kuten esimerkistä näkyy, Ajax-kutsu <iron-ajax>-elementtiä käyttämällä on todella yksinkertaista eikä suuria koodimääriä tarvitse kirjoittaa. [60.]

<dom-repeat>

Tässä esimerkissä esitellään edellisessä esimerkissä saadun datan esittäminen <dom-repeat>-elementin avulla. <dom-repeat>-elementin avulla on mahdollista käydä datasta koostuvaa taulukkoa (array) läpi ja tallentaa tieto muuttujaan. Koodiesimerkissä 14 on esitelty Foursquaresta saadun datan läpikäyminen <dom-repeat>-elementin avulla.

```
1.  <template is="dom-repeat" items="[[data.response.categories]]">
2.    <span>{{item.name}}</span>
3.    <span>{{item.id}}</span>
4.  </template>
```

Koodiesimerkki 14. <dom-repeat>-elementin käyttö datan läpikäymisessä.

Alussa määritellään sivupohja noudattamaan <dom-repeat>-toiminnallisuutta, minkä jälkeen ’items’-attribuutissa määritellään, mitä taulukkoa käydään läpi. Sivupohjan sisällä voidaan tulostaa taulukosta arvoja käskyllä ’item.*’. Tämä tekniikka on hyödyllinen taulukkojen läpikäymiseen, ja syntaksi on yksinkertainen ja helppo toteuttaa. Varsinaisessa sovelluksessa <dom-repeat>-elementtiä käytettiin Foursquaresta saadun datan läpikäymiseen ja listaukseen. [61.]

<app-route> ja <app-location>

<app-route>-elementti on tarkoitettu reitityksen toteuttamiseen verkkosovelluksessa, ja sitä yleensä käytetään yhdessä <app-location>-elementin kanssa. Tyypillisessä tapauksessa elementti tuottaa olion, joka saa jonkin arvon tallennettuna ’route’-ominaisuuteen kuvaamaan reitin tilaa. Sovelluksessa elementtiä käytettiin näkymien vaihtamiseen käyttäjän navigoidessa sovelluksessa. [62.] Koodiesimerkissä 15 on esimerkki <app-route>- ja <app-location>-elementtien käytöstä.

```
1.  <link rel="import" href="../bower_components/app-route/app-location.html">
2.  <link rel="import" href="../bower_components/app-route/app-route.html">
3.  <app-location route="{{route}}"></app-location>
4.  <app-route
5.    route="{{route}}"
6.    pattern="/:page"
7.    data="{{routeData}}"
8.    tail="{{subroute}}">
9.  </app-route>
10. //Parametrien määritys Polymer() -funktion sisällä
11.   page: {
12.     type: String,
13.     reflectToAttribute: true,
14.     observer: '_pageChanged',
15.   }
```

Koodiesimerkki 15. <app-route>- ja <app-location>-elementtien käyttö.

<app-location>-elementti mahdollistaa synkronoinnin selaimen osoitekentän ja sovelluksen tilan välillä. Elementissä määritetään olio ’route’, joka päivittyy joka kerta, kun sovelluksen tila muuttuu. <app-route>-elementti hyödyntää ’route’-oliota ja asettaa sen ensimmäisen attribuutin arvoksi rivillä 3. ’pattern’-attribuutissa ’/:page’ asetetaan seuraavana olevaan ’data’-olioon. ’data’ sisältää parametriset arvot, jotka ’route’ on tuottanut, ja ’tail’ sisältää kaikki muut polut, joita ’pattern’ ei määrittänyt. Polymer()-funktion sisällä ’page’-ominaisuus määritellään. ’reflectToAttribute’ määrää sen, että muutoksen tapahtuessa koko ’page’ päivittyy, ja ’observer’ kutsuu _pageChanged() -funktiota, jossa sovelluksen reititys päivittyy. <app-route>-elementtiä käyttämällä sovelluksen reititys on yksinkertaista toteuttaa. [62.]

<iron-pages>

Kaikessa yksinkertaisuudessaan <iron-pages>-elementin avulla määritetään, mikä sen lapsielementeistä näytetään käyttäjälle. Koodiesimerkissä 16 on esitettynä <iron-pages>-elementin käyttöä näkymän vaihtoon sovelluksessa.

```
1.  <link rel="import" href="../bower_components/iron-pages/iron-pages.html">
2.  <iron-pages
3.    selected="[[routeData.page]]"
4.    attr-for-selected="id"
5.    fallback-selection="map">
6.    <my-map id="map" data="{{dataCollection}}"></my-map>
7.    <my-venuelist id="venuelist" data="{{dataCollection}}"></my-venuelist>
8.  </iron-pages>
```

Koodiesimerkki 16. <iron-pages>-elementin käyttö.

’selected’-attribuutti hakee ja asettaa valitun näkymän käyttäjälle näkyviin. Tässä esimerkissä aiemmin <app-route>-elementissä reititysdata on sidottu ’routeDataan’, jolloin asettamalla ’routeData.page’ attribuutin arvoksi saadaan ’page’:n sisältö näkyviin. ’attr-for-selected’-attribuutti määrittää, millä tunnisteella haluttu näkymä valitaan. Esimerkissä on valittu ’id’, mutta attribuutin arvoksi voi antaa vaihtoehtoisesti esimerkiksi ’name’:n. ’fallback-selection’ määrittää, mikä näkymä ladataan, jos jokin tarkoitetun näkymän latauksessa menee vikaan. <iron-pages>-elementin sisällä määritetään näkymät, joita halutaan sisällyttää sovellukseen. Esimerkissä on kaksi omaa elementtiä, jotka edustavat sovelluksen eri näkymiä. Toinen elementti sisältää karttanäkymän ja toinen listanäkymän. Käyttäjän navigoidessa näkymien välillä <iron-pages>-elementti huolehtii oikean näkymän näyttämisestä. [63.]

<google-map> ja <geo-location>

Miljoonat verkkosivut hyödyntävät Google Maps -ohjelmointirajapintoja karttojen ja sijaintien esittämiseen. Suosio varmasti johtuu ainakin osittain ohjelmointirajapinnan helppokäyttöisyydestä. Google on tehnyt karttojen esittämisestä webkomponenttien käyttäjille vieläkin helpompaa, sillä se on luonut <google-map>-elementin. Elementin avulla sivustolla voidaan esittää haluttu kartta. <geo-location>-elementti on yksityisen kehittäjän tekemä, ja myös sen saa ladattua webcomponents.org-sivustolta. Elementti perustuu Geolocation-ohjelmointirajapintaan (https://developer.mozilla.org/en-US/docs/Web/API/Geolocation), ja sen tarkoituksena on paikantaa käyttäjä. Näitä kahta elementtiä käyttämällä koodiesimerkissä 17 on esitetty, miten käyttäjä paikannetaan ja miten hänen sijaintinsa saadaan Googlen kartalle näkyviin. Lisäksi <google-map-marker>-elementtiä hyödyntämällä asetetaan osoitin käyttäjän sijainnin kohdalle. [64; 65.]

```
1. <link     rel="import"     href="../bower_components/geo-location/geo-loca-
   tion.html">
2. <link rel="import" href="../bower_components/google-map/google-map.html">
3. <dom-module id="my-map">
4.   <template>
5.     <style>
6.       google-map{
7.         display: block;
8.         height: 100%;
9.       }
10.    </style>
11.    <geo-location watch-pos latitude="{{ latitude }}" longitude="{{ longi-
   tude }}"></geo-location>
12.    <google-map id="map" latitude="[[latitude]]" longitude="[[longitude]]"
   zoom="13" api-key="XXXXXXX">
13.      <google-map-marker latitude="[[latitude]]" longitude="[[longitude]]"
   draggable="false"></google-map-marker>
14.    </google-map>
15. </template>
16. </dom-module>
```

Koodiesimerkki 17. Käyttäjän paikannus ja kartan esittäminen <geo-location>- ja <google-map>-elementtien avulla.

Aluksi <google-map>- ja <geo-location>-elementit ladattiin Bowerin avulla, ja ne otettiin tuonti-HTML-tekniikalla käyttöön. Jotta kartta saataisiin näkyviin sivustolle, sille on määritettävä korkeus. Rivillä 6 elementille on annettu halutut tyylimääritteet, ja rivillä 11 <geo-location>-elementti määritellään. Kun ’watch-pos’ on asetettu attribuutiksi, elementin ’latitude’ ja ’longitude’ päivittyvät aina laitteen koordinaattien muuttuessa. ’latitude’- ja ’longitude’-attribuutit saavat arvoikseen käyttäjän senhetkiset sijaintikoordinaatit. <google-map>-elementin avulla kartta esitetään sivustolla. Tämäkin elementti voi saada useita eri ominaisuuksia, ja kaikki mahdollisuudet on listattu osoitteessa https://www.webcomponents.org/element/GoogleWebComponents/google-map/google-map. Esimerkissä elementille on annettu ’latitude’ ja ’longitude’ keskipisteen määrittämiseksi. Keskipisteen koordinaatit saadaan datakytkennän avulla <geo-location>-elementiltä hakasulkuja käyttämällä, niin kuin rivillä 12 on esitetty. ’zoom’ on tarkoitettu zoomauksen määrittämiseksi. Näiden attribuuttien lisäksi on määritetty ’api-key’, joka on räätälöity jokaiselle kehittäjälle ja jonka saa Google Maps -ohjelmointirajapintojen dokumentaatiota seuraamalla. Lopuksi vielä käytetään <google-map-marker>-elementtiä osoittimen lisäämiseksi kartalle.

Myös tässä elementissä hyödynnetään <geo-location>-elementin luomia sijaintikoordinaatteja. [64; 65.]

Edellä kuvatut esimerkit koostavat pääpiirteet toteutetulle sovellukselle ja sen näkymille. <iron-ajax>- ja <dom-repeat>-elementtejä käytetään sovelluksessa datan hakemiseen ja esittämiseen, <app-route>:lla toteutettiin sovelluksen reititys, <iron-pages>-elementti huolehti sovelluksen näkymien näyttämisestä ja <google-map>- ja <geo-location>-elementit loivat sovellukseen kartan ja sen toiminnallisuudet.

Insinöörityönä toteutettu sovellus perustuu käyttäjän sijaintitietoihin ja häntä lähellä olevien paikkojen etsimiseen. Sovelluksen tarkoituksena on paikantaa käyttäjä, näyttää käyttäjän sijainti kartalla ja etsiä käyttäjän läheltä ravintoloita ja kahviloita. Käyttäjä pystyy rajaamaan paikkoja suodattamalla hakuetäisyyttä ja tuloksien lukumäärää. Toteutin sovelluksen yksisivuisena sovelluksena (Single-page application), ja ulkoasun pohjana käytin Polymerin App-layoutia (https://www.webcomponents.org/element/PolymerElements/app-layout), joka koostuu erilaisista ulkoasuelementeistä.

4.3 My Food 2.0 -verkkosovellus Polymerilla

Digitaaliset kartat ja karttojen esittäminen verkkosovelluksissa ovat kasvattaneet suosiotaan viime vuosina. Suosio on ymmärrettävää, koska kartat voivat olla todella informatiivisia ja käytännöllisiä. Google Maps on edelleen johtavassa asemassa, kun puhutaan digitaalisista kartoista, mutta yhä enemmän myös muut yritykset ovat alkaneet kehittää vaihtoehtoja Google Mapsille. Monet digitaalisten karttojen kehittäjät ovat tuottaneet myös ohjelmointirajapintoja, joita muun muassa sovellusten kehittäjät voivat hyödyntää. Monet karttojen ohjelmointirajapinnat tuovat samanlaisia ominaisuuksia kehittäjille kuin muutkin, mutta myös uniikkeja ominaisuuksia on olemassa. [66.]

Insinöörityötä tehdessä ensimmäinen iso päätös oli kartan ja karttaohjelmointirajapinnan valitseminen, koska kartta toimisi päänäkymänä käyttäjälle ja se sisältäisi päätoiminnot. Työn aikana testattiin kolmea eri karttaohjelmointirajapintaa sovellukseen, ja niistä lopulta valittiin yksi varsinaiseen sovellukseen. Ohjelmointirajapintoja olivat Google Maps, Here ja Mapbox. Nämä kolme valittiin niiden suosion perusteella. Heti aluksi Here karsiutui pois, koska sen käyttö oli rajattu vain kolmeen kuukauteen, jonka jälkeen palvelusta tulisi maksullinen. Sekä Mapboxista että Google Mapsista on kehitetty räätälöidyt

elementit: <mapbox-gl> ja <google-map>. Vaikka Mapboxin tuottama kartta oli visuaalisesti näyttävämpi ja hyvää vaihtelua Googlen kartalle, Google Maps vei voiton tässä vertailussa. Suurimpia syitä Googlen kartan valintaan oli sen helppokäyttöisyys. Vaikka Mapboxistakin on olemassa räätälöity elementti kartan esittämiseen, sen lisäksi tarvittiin suuri määrä JavaScript-koodia yksinkertaisen osoittimen lisäämiseen kartalle. Google Mapsin räätälöidyn elementin lisäksi myös osoittimille on kehitetty räätälöity elementti <google-map-marker>, jonka avulla osoittimen sai lisättyä karttaan kirjoittamalla yhden rivin koodia. Lisäksi Mapboxin kartta ei aina renderöitynyt oikein sivulle, mikä vaikutti myös suuresti päätöksen tekoon.

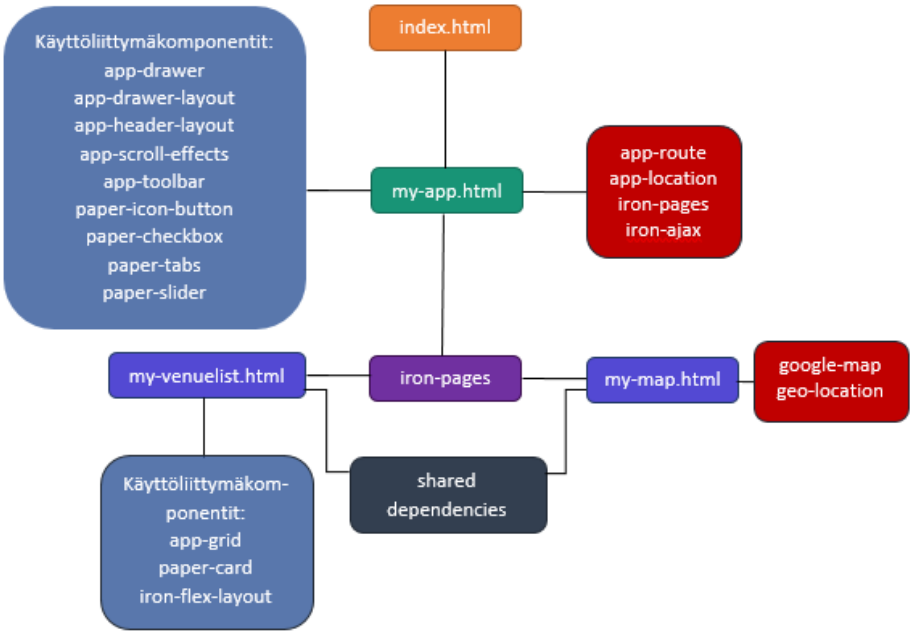
Kartan lisäksi sovelluksen toteutukseen tarvittiin sijaintiin perustuva paikkatietoja välittävä ohjelmointirajapinta. Myös tässä kohtaa vaihtoehtoja oli useampia kuin yksi, mutta aikaisemman kokemuksen vuoksi ohjelmointirajapinnaksi valittiin Foursquare. Foursquare on suosituin sijaintiin perustuva sosiaaliseen interaktioon tarkoitettu sovellus, jonka avulla käyttäjät voivat muun muassa etsiä heitä kiinnostavia paikkoja, kuten ravintoloita tai kulttuurikohteita [67]. Foursquaren ohjelmointirajapinnan avulla kehittäjät voivat hakea tietoja paikoista ja käyttäjistä. Insinöörityössä hyödynnettiin Foursquaren ’venues’-toimintoa, jonka avulla saatiin tietoja käyttäjän lähellä olevista paikoista.

Ohjelmointirajapintojen valinnan jälkeen sovelluksen kehittäminen lähti liikkeelle sen arkkitehtuurin ja rakenteen suunnittelusta. Lopputulos koostui kahdesta näkymästä, joiden välillä käyttäjä voi navigoida: karttanäkymä ja listanäkymä. Sovellus koostuu kolmesta luodusta räätälöidystä elementistä: <my-app>, <my-map> ja <my-venuelist>. <my-app>-elementtiin sisällytettiin sovelluksen yleinen toimintalogiikka, ulkoasun toteutus, reititys ja Ajax-kutsut Foursquaren ohjelmointirajapinnalle. Kaksi muuta räätälöityä elementtiä sisälsivät sovelluksen näkymät, eli kartan ja listan. Sovelluksen ulkoasussa hyödynnettiin app-layoutin elementtejä, ja lisäksi sovelluksen toteutuksessa hyödynnettiin Polymer-tiimin kehittämää App Toolboxia, joka on kokoelma komponentteja ja työkaluja verkkosovelluksen kehitykseen.

App layout -elementit tuovat kehittäjille työkaluja, joilla saadaan helposti toteutettua responsiivisia käyttöliittymiä sovelluksille. App layout -elementtien avulla sovelluksesta saatiin helposti responsiivinen ja ulkoasultaan toimiva. Muun muassa yläpalkki, sivupalkki ja listanäkymän lista on toteutettu App layout -elementtien avulla (<app-header>, <app-drawer>, <app-grid>).

Polymerin Toolbox (https://www.polymer-project.org/1.0/toolbox/index) on valmis paketti komponentteja ja työkaluja verkkosovelluksen kehitykseen, ja sen perustana on komponenttipohjainen arkkitehtuuri. Toolbox sisältää valmiiksi työkaluja muun muassa verkkosovelluksen reititykseen ja responsiivisen käyttöliittymän toteutukseen. Toolbox noudattaa PRPL-mallia (Push, Render, Pre-cache, Lazy-load), jotta sovellus toimisi mahdollisimman optimaalisesti. Toteutetun sovelluksen pohjana käytettiin Toolboxia, koska sitä kautta sovellukselle saatiin hyvä pohja, josta sitä lähdettiin kehittämään.

Kuvassa 11 on havainnollistettu sovelluksen arkkitehtuuria, ja siinä on esitettynä kaikki valmiit räätälöidyt elementit, joita sovelluksessa hyödynnettiin. Valmiin sovelluksen koodeja voi tarkastella osoitteessa https://github.com/hannaa/lopputyo/tree/master/my-app.

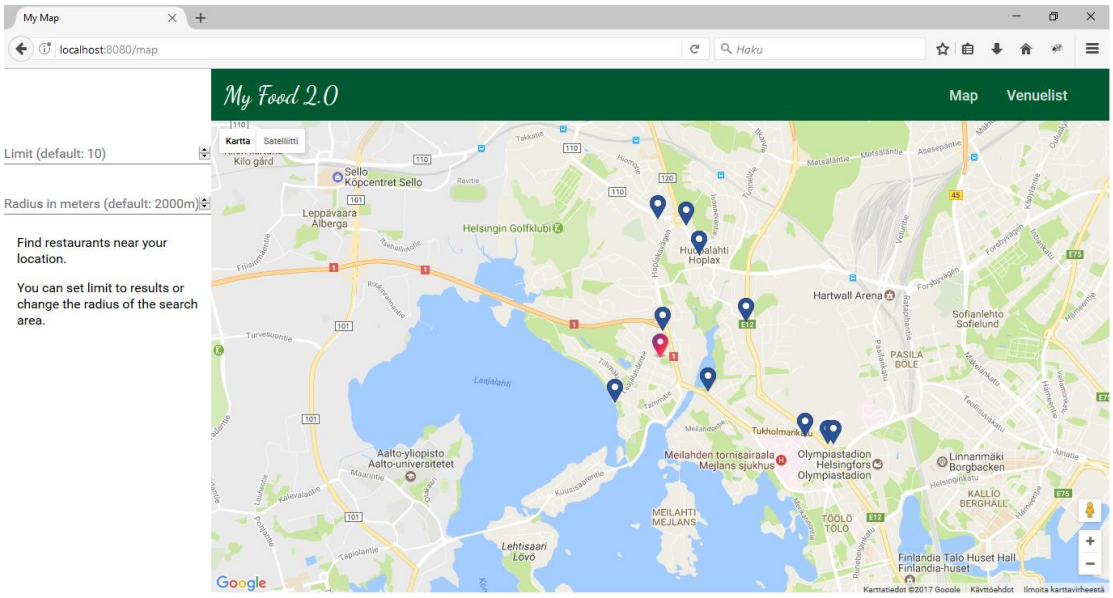


Kuva 11. My Food 2.0 -verkkosovelluksen arkkitehtuuri.

Sovelluksessa index.html toimi niin sanottuna sisääntulopisteenä (entry-point), joka sisältää vain välttämättömät asiat, kuten webcomponents.js-liitännäisen ja <my-app>-elementin liittämisen. <my-app>-elementti oli ensimmäinen räätälöity elementti, ja se toimi ikään kuin runkona (shell) sovellukselle. Sovelluksen toimintalogiikka, kuten reititys ja näkymien vaihdot, sekä kaikkia näkymiä koskevat yhteiset käyttöliittymäkomponentit sisältyivät <my-app>-elementtiin. Niin sanottuina sovelluksen palasina (fragment) toimivat toisistaan riippumattomat näkymät: <my-venuelist> ja <my-map>. Karttaelementti sisältää toiminnallisuudet ja elementit Googlen kartan lataamiseen ja käyttäjän paikantami-

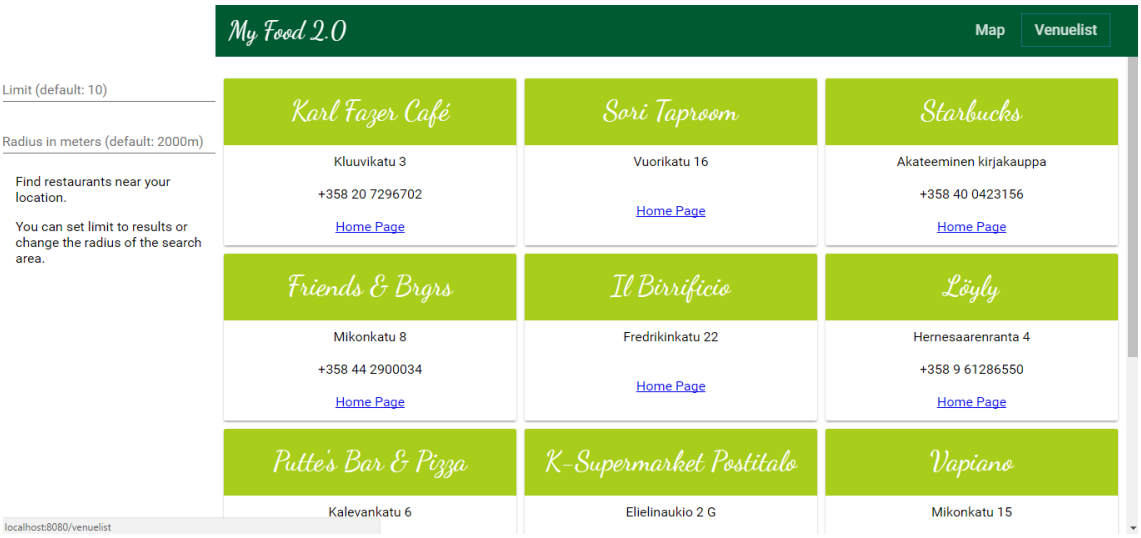
seen. Listanäkymä sisältää käyttöliittymäelementit listan esittämiseen käyttäjälle. Näkymien lataukseen käytetään ’lazy-load’-menetelmää, joka tarkoittaa sitä, että näkymä ladataan vasta, kun käyttäjä on siihen navigoinut. Tällä tavalla sovelluksen lataus nopeutuu, kun osat ladataan vasta, kun niitä tarvitaan.

Ensimmäinen ja oletuksena toimiva näkymä on karttanäkymä, jossa käyttäjä näkee oman sijaintinsa kartalta. Tämän lisäksi käyttäjä näkee kymmenen itseään lähinnä olevan ravintolan tai kahvilan sijainnit. Näkymä on esitetty kuvassa 12.



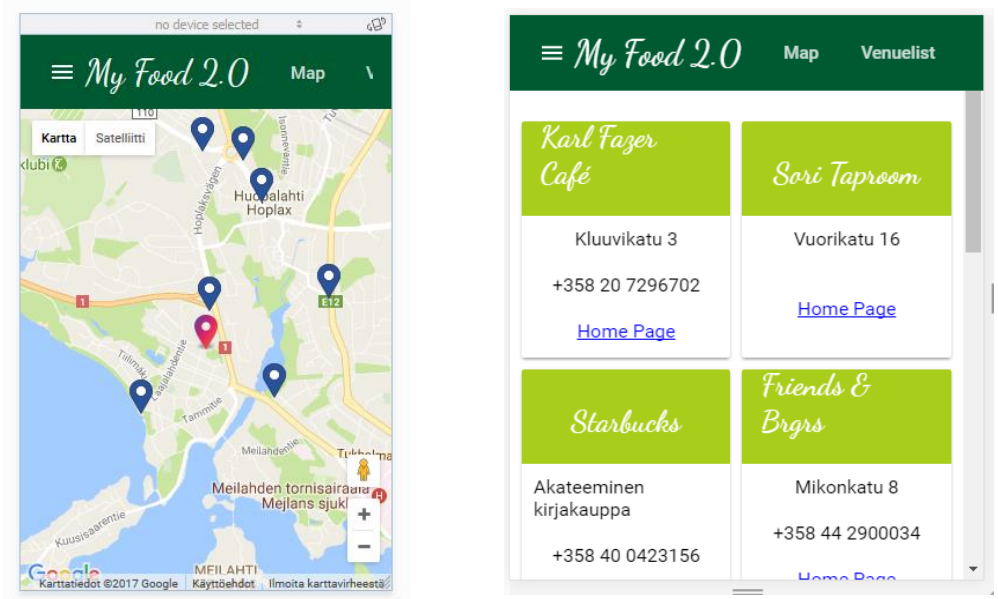
Kuva 12. Sovelluksen karttanäkymä.

Kuvassa 13 on esitetty listanäkymä, jossa kartalla olevat ravintolat ja kahvilat ovat listattuna.



Kuva 13. Sovelluksen listanäkymä, jossa on listattuna ravintolat ja kahvilat.

Koska oletuksena on, että käyttäjä etsii itselleen sopivaa ravintolaa tai kahvilaa ollessaan tien päällä ja käyttää sovellusta mobiililaitteellaan, sovelluksesta tehtiin responsiivinen, jolloin se mukautuu myös mobiililaitteilla käytettäväksi. Kuvassa 14 on esitetty molemmat näkymät mobiililaitteella.



Kuva 14. Sovelluksen näkymät mobiililaitteella.

Listanäkymässä jokaisesta ravintolapaikasta on listattuna yhteystiedot sekä linkit kotisivuille. Sovelluksen yläpalkissa on linkit näkymien vaihtoon. Sovelluksessa on myös mahdollista suodattaa hakutuloksia hieman. Käyttäjä voi valita, montako paikkaa näytetään kartalla. Minimi on 1 ja maksimi 50. Tämän lisäksi käyttäjä voi määrittää, kuinka suurelta alueelta paikkoja etsitään. Käyttäjän muokatessa suodattimia sovellus lähettää aina uuden Ajax-kutsun Foursquaren ohjelmointirajapinnalle, minkä jälkeen tulokset päivittyvät karttaan ja listaan. Tuloksia voi suodattaa sivupalkissa olevista liukusäätimistä (slider). Mobiililaitteilla sivupalkin saa esiin painamalla menu-nappia.

5 My Food 2.0 -sovelluksen tulokset ja ongelmat

Insinöörityön tarkoituksena oli selvittää, mitä uutta webkomponenttien käyttö tuo verkkosovelluksiin ja mitä hyötyä niistä on. Työssä testattiin Polymer-kirjaston käyttöä ja sen tarjoamia ominaisuuksia verkkosovelluskehitykseen. Tässä luvussa on tarkoitus hieman analysoida työn tuloksia; mitä hyviä ja huonoja puolia webkomponenttien käytöstä on ja

mitä hyötyjä Polymer-kirjasto tarjosi mielestäni sovelluksen kehitykseen. Myös työssä ilmenneitä ongelmia esitellään. Lisäksi pohditaan webkomponenttien mahdollisuuksia tulevaisuuden verkkosovelluksissa.

Webkomponenttien hyödyt

Webkomponenttien tavoitteena on tuoda modulaarisuutta verkkosovelluksiin ja antaa kehittäjille mahdollisuus luoda ominaisuuksiltaan uniikkeja elementtejä. Kehittäjän näkökulmasta webkomponentit tuovat paljonkin hyviä ominaisuuksia verkkosovelluksien kehitykseen. Elementtien rakentaminen toiminnallisuuksien mukaan (<my-app>, <my-map>, <my-venuelist>) mahdollisti modulaarisemman ja selkeämmän rakenteen sovellukselle. Tyylien ja toiminnallisuuksien kapselointi onnistui sivupohjien ja varjo-DOMin avulla, ja tuonti-HTML:n avulla räätälöidyt elementit voitiin liittää yhteen.

Suunnitteluvaiheessa sovelluksen arkkitehtuuri hahmottui nopeasti, koska räätälöityjä elementtejä rakentamalla tarvittavat toiminnallisuudet voitiin jakaa selkeiksi erillisiksi osioiksi. Kartan sisältämä elementti sisälsi vain kartan tyylit ja toiminnallisuudet ja listan sisältämä elementti listan ominaisuudet sekä my-app yleiset toiminallisuudet. Olin aiemmin toteuttanut samankaltaisen sovelluksen jQuerylla, ja siinä hankaluuksia tuotti juuri toiminnallisuuksien erottelu selkeisiin osioihin. Koko insinöörityöprosessin ajan koodi pysyi selkeänä, ja koko ajan tiedettiin, mihin mikäkin osa vaikuttaa. Lopputuloksena oli neljä HTML-tiedostoa: index.html, my-app.html, my-map.html ja my-venuelist.html, jotka saatiin liitettyä yhteen tuonti-HTML:n avulla.

Sivupohjia ja varjo-DOMia käyttämällä elementtien tyylit ja toiminnallisuudet saatiin kapseloitua koskemaan vain kyseistä elementtiä, jolloin esimerkiksi tyylimäärittelyt eivät päässeet vaikuttamaan muihin elementteihin. Oman kokemukseni perusteella tyylimäärittelyt usein voivat helpostikin vaikuttaa myös sellaisiin osiin, joihin niiden ei kuuluisi vaikuttaa, ja välillä tämän välttäminen voi olla kovan työn takana. Webkomponenteissa sivupohjat ja varsinkin varjo-DOM tarjosivat toimivan ratkaisun tähän ongelmaan.

Polymer.js oli mielestäni jälkeenpäin ajateltuna oikea valinta työn toteuttamiseen. Koska teknologiat ja kirjastot olivat aivan uusia minulle, Polymerin dokumentaatio, esimerkit ja tutoriaalit auttoivat pääsemään alkuun nopeasti. Parhaat puolet Polymerin käytössä oli-

vat muun muassa kaksisuuntainen datakytkentä, valmiit räätälöidyt elementit, App Toolbox ja varjoisa DOM. Toki myös joitakin huonoja puolia oli, ja myös kehitettävää ilmeni työn aikana.

App Toolbox ja Polymer CLI tarjosivat erinomaisen pohjan sovelluksen kehitykselle. App Toolboxin tarjoamat työkalut, kuten muun muassa reititys, auttoivat sovelluksen peruspilarien rakentamisessa. Kun perusasiat olivat sovelluksessa kunnossa, oli helppo lähteä rakentamaan muita ominaisuuksia ja räätälöityjä elementtejä. Polymer CLI:n käyttö mahdollisti tehtyjen muutoksien seuraamisen helposti. Polymer.js sisältää oletuksena varjoisan DOMin liitännäisen varjo-DOMin sijaan, jolloin kirjaston koko ei paisu liian suureksi eikä sivun latausnopeus kasva turhan suureksi.

Valmiit räätälöidyt elementit, joita Polymerin tiimi ja muut kehittäjät ovat kehittäneet, toivat paljon helpotusta oman sovelluksen rakentamiseen. Esimerkiksi app-layout-elementtien avulla responsiivisen ja visuaalisesti toimivan käyttöliittymän toteuttaminen oli yksinkertaista ja nopeaa. Jos näitä elementtejä ei olisi ollut valmiina, olisi aikaa kulunut huomattavasti kauemmin. Tärkeän lisän toivat myös eri toiminnallisuuksia sisältävät elementit, kuten <app-route> ja <iron-ajax>. Reititys saatiin sovellukseen vähällä vaivalla, ja Ajax-kutsujen toteuttaminen <iron-ajax>-elementtiä käyttämällä sujui vaivattomasti. Yksi tärkeimmistä elementeistä oli <google-map>, koska sen avulla sovelluksen yksi päätoiminnallisuus saatiin toteutettua. Ilman valmista karttaelementtiä olisi pitänyt kirjoittaa huomattavasti enemmän JavaScriptiä saman lopputuloksen saavuttamiseksi.

Hyödyllisin ominaisuus mielestäni Polymerin käytössä oli kaksisuuntainen datakytkentä, joka mahdollisti tiedon jakamisen eri elementtien välillä helpolla tavalla. Sovelluksessa datakytkentää hyödynnettiin paikkatietojen välittämisessä sekä karttaelementille että listaelementille. Ajax-kutsun jälkeen paikkatiedot listattiin taulukkoon, joka jaettiin datakytkennän avulla muihin elementteihin. Tällä tavalla paikat voitiin esittää kartalla ja listalla. Ilman datakytkentää tiedon jakaminen elementtien välillä olisi ollut huomattavasti haastavampaa, vaikka ominaisuuden opetteleminen veikin hieman aikaa.

Haasteet

Suurin ongelma webkomponenttien käytössä myös tässä projektissa oli selaintuen puute osassa selaimista. Tällä hetkellä vain Googlen Chrome ja Opera tukevat kaikkia

webkomponentti-rajapintoja, joten liitännäisten käyttö oli välttämätöntä. webcomponents.js-liitännäinen tuo webkomponenttien tuen kaikkiin selaimiin, mutta se kasvattaa sivun latausnopeutta jonkin verran. Työssä hyödynnettiin webcomponents-lite.js:ää, jossa siis varjo-DOMin tilalla on tuettuna varjoisa DOM, jolloin liitännäisen koko pieneni huomattavasti. webcomponents.js:n koko on 116 kB, kun taas webcomponents-lite.js:n koko on vain 39,9 kB. Pienempää liitännäistä käyttämällä myös sivun latausnopeus säilyi jokseenkin hyvänä.

Huonona puolena Polymerissa on kirjaston koko. Suuren koon vuoksi latausnopeus kasvaa, ja nykyään latausnopeus on ratkaisevassa asemassa käyttäjän näkökulmasta. Mitä kauemmin sivusto latautuu, sitä todennäköisemmin käyttäjä luovuttaa ja jättää sivuston ennen latauksen valmistumista. Yksisivuisissa sovelluksissa Polymer on kokonsa puolesta vielä siedettävä, mutta useampisivuisiin sovelluksiin siirryttäessä koosta alkaa tulla ongelma, koska sisältöä ladataan enemmän ja useammin. Koska toteutettu sovellus oli yksisivuinen, latausnopeus pysyi siedettävänä.

Toinen ongelma projektin aikana oli tuonti-HTML:ien ja niiden riippuvuuksien lataaminen. Koska sovelluksessa käytettiin noin 20:tä eri elementtiä, sovelluksen latausnopeus kasvoi suhteellisen suureksi. Keskimääräinen latausnopeus sivulla oli noin 4 sekuntia, mikä on optimaalisen latausnopeuden ylärajoilla.

Yleisiä ongelmia sovelluksen kehityksessä tuli vastaan reitityksessä ja Foursquaren kanssa. Jostain syystä Ajax-kutsuja Foursquarelle lähetettäessä pyyntö lähti kahteen kertaan. Toinen kyselyistä palautti oikeat tulokset ja toinen errorin, jossa sanottiin, että käyttäjätunnukset eivät ole oikein. Kuvassa 15 nähdään molemmat kyselyt.

Kuva 15. Ajax-kyselyiden tilanne selaimen konsolissa.

Ylin punaisella merkitty kysely tuotti virheilmoituksen ja viimeisen rivin kysely oikeat tulokset. Työn aikana yritin useaan kertaan selvittää, mistä ongelma johtuu, siinä kuitenkaan onnistumatta.

Toinen ongelma ilmeni reitityksen toteutuksessa siirrettäessä sovellusta Polymer CLI:n palvelimelta GitHubin palvelimelle. Sivustolle siirryttäessä päänäkymä, eli kartta, ei latau-du, ennen kuin navigointipalkista painetaan ’Map’-nappia. Kuvassa 16 on näkymä sivustolle mentäessä.

Kuva 16. Sovelluksen näkymä GitHub -palvelimella.

Selaimen konsoliin tuli virheilmoituksia reitityksestä, ja sovelluksen näkymien sisältö latautui vasta käyttäjän toimien perusteella. Nämä kaksi ongelmaa eivät mielestäni kuitenkaan johtuneet webkomponenteista tai Polymerista, vaan jostain syntaksivirheestä tai ohjelmointirajapintojen toimimattomuudesta.

Webkomponenttien tulevaisuus

Yleisesti ottaen tämän työn perusteella webkomponentit ja Polymer voivat hyvinkin olla tulevaisuudessa isoja tekijöitä verkkosovellusten kehityksessä. Ne tarjoavat sovelluskehitykseen modulaarisuutta ja selkeyttä, ja räätälöityjen elementtien luominen mahdollistaa tyylien ja toiminnallisuuksien kapseloinnin toimiviksi kokonaisuuksiksi.

Kunhan muutamasta kompastuskivestä päästään eroon, uskon, että näiden teknologioiden suosio kasvaa. On selvää, että webkomponenttien suurin ongelma on selaintuen puute suurimmassa osassa selaimista. Tähän toivotaan lähivuosina muutosta. Applen ja Microsoftin edustajat ovat sanoneet, että webkomponentti-teknologiat ovat mielenkiintoisia, ja heti kun he ovat tyytyväisiä teknologioiden määritelmiin, niiden tuen lisäämistä selaimiin voidaan vakavasti harkita [27]. Tätä ennen kuitenkin liitännäisten käyttäminen on melkeinpä pakollista.

Polymer.js:ää kehitetään koko ajan, ja tämän insinöörityön teon aikana julkaistiin Polymer 2.0. Uudessa versiossa varjoisa DOM on erotettu erilliseksi liitännäiseksi, kirjaston käyttäminen muiden kirjastojen ja ohjelmistokehysten kanssa on tehty helpommaksi ja datakytkentää on paranneltu entisestään. Polymer 2.0:n syntaksikin eroaa joissain määrin Polymer 1.x:n syntaksista. Polymerin tiimi on luonut kehittäjille ohjeiston, jota noudattamalla kehittäjät voivat halutessaan vaihtaa 1.x:stä 2.0:aan.

Webkomponentit ja Polymer toivat sovelluskehitykseen juuri niitä ominaisuuksia, mitä oletinkin etukäteen: modulaarisuutta, selkeyttä ja erillisien komponenttien tuomia hyötyjä. HTML:n ongelmana on ollut, että olemassa olevia elementtejä ei ole voinut muokata tai laajentaa, ja CSS:n ongelmana on ollut tyylien vuotaminen muihin elementteihin. Webkomponentit tuovat standardoidun ratkaisun muun muassa näihin ongelmiin. Uskon, että tulevaisuudessa webkomponenteista ja Polymerista kasvaa varteenotettavia kilpailijoita nykypäivän suosituimmille ohjelmistokehyksille.

6 Yhteenveto

Insinöörityössä tutkittiin, mitä webkomponentit ovat ja mitä hyötyjä ne tuovat verkkosovelluskehitykseen. Tarkemmassa tarkastelussa oli Googlen Polymer-kirjasto, jota käyttämällä toteutettiin yksinkertainen verkkosovellus.

Internet on vuosien varrella kehittynyt nopeasti, ja samalla on teknologioiden määrä kasvanut paljon. Viimeisen kymmenen vuoden aikana sovelluskehitysmaailmassa erilaisten kirjastojen ja ohjelmistokehysten määrä on kasvanut kymmeniin tuhansiin. Viime vuosina verkkosovellusten modulaarisuus ja erillisten komponenttien käyttö on kasvattanut suosiotaan. Modulaarisuuteen liittyy kuitenkin ongelmia, joihin on kaivattu standardoitua ratkaisua, joka määrittäisi tavan kehittää uudelleen käytettäviä erillisiä komponentteja. W3C on alkanut kehittää standardoituja teknologioita, webkomponentteja, näiden ongelmien ratkaisemiseen.

Webkomponentit koostuvat neljästä eri teknologiasta, joista jokainen tuo hyödyllisiä ominaisuuksia sovelluskehitykseen. Räätälöidyt elementit mahdollistavat omien HTML-elementtien rakentamisen. Sivupohjat ja varjo-DOM mahdollistavat tyylien ja toiminnallisuuksien kapseloinnin koskemaan vain tiettyä elementtiä, ja tuonti-HTML tarjoaa helpon

tavan liittää elementtejä yhteen. Koska webkomponentit ovat kuitenkin vasta kehitteillä, niihin liittyy myös ongelmia.

Webkomponenttien suurin ongelma on selaintuen puute suurimmassa osassa selaimista. Tällä hetkellä vain Googlen Chrome- ja Opera-selaimet tukevat kaikkia webkomponentteja. Ratkaisuksi on kuitenkin kehitetty liitännäisiä, jotta webkomponentit saadaan toimimaan kaikissa moderneissa selaimissa. Webkomponenttien kehitykseen on kehitetty myös muutamia kirjastoja, muun muassa Googlen Polymer, joka toimii kevyenä kerroksena webkomponenttien päällä yhdistäen kaikkien neljän teknologian ohjelmointirajapinnat toisiinsa. Polymer-kirjasto tarjoaa hyvät eväät verkkosovelluksen rakennukselle webkomponentteja hyödyntäen.

Webkomponenttien isoin ongelma on selaintuen puute. Jos muut selaimet eivät lähivuosina lisää webkomponenttien tukea selaimiin, voi olla, että webkomponentit eivät koskaan pääse näyttämään todellista hyötyään suurelle yleisölle. Tällä hetkellä webkomponenttien tulevaisuus on kuitenkin valoisa, ja Google panostaa myös tulevaisuudessa Polymerin kehitykseen. Siitä onkin julkaistu jo uusi versio, Polymer 2.0.

Sovelluksen toteuttaminen Polymerilla oli kokonaisuudessaan yksinkertaista kirjastoon tutustumisen jälkeen. Koodia oli helppo kirjoittaa ja lukea. Polymer-tiimi tarjosi myös hyviä työkaluja sovelluksen rakentamiseen, ja niiden avulla oli helppo lähteä liikkeelle, vaikka teknologiat ja kirjasto olivat aivan uusia. Myös Polymerin yhteisön laajuus toi helpotusta sovelluksen kehitykseen ja tiedon hakuun.

Insinöörityö onnistui kuta kuinkin suunnitelmien mukaan. Aikataulussa pysyminen tuotti hieman vaikeuksia, mutta työ saatiin tehtyä ennen aikarajaa. Työ saavutti sille asetetut tavoitteet hyvin, ja kokonaisuudessaan prosessi suunnittelusta viimeistelyyn sujui loppujen lopuksi odotetusti.

Lähteet

1 Connors, Adam. 2010. Mobile Web Application Best Practices: W3C Recommendation. Verkkodokumentti. W3C. <https://www.w3.org/TR/mwabp/#webapp-defined>. Luettu 27.1.2017.

2 Usage of Server-Side Programming Languages for Websites. 2017. Verkkodokumentti. W3Techs. <https://w3techs.com/technologies/overview/programming_language/all>. 27.1.2017. Luettu 27.1.2017.

3 Usage of Client-Side Programming Languages for Websites. 2017. Verkkodokumentti. W3Techs. <https://w3techs.com/technologies/overview/client_side_language/all>. 27.1.2017. Luettu 27.1.2017.

4 From History of Web Application Development. 2016. Verkkodokumentti. Devsaran. <https://www.devsaran.com/blog/history-web-application-development>. 8.7.2016. Luettu 27.1.2017.

5 Kukovec, Darko. 2014. Web Components – Building Blocks of the Future Web. Verkkodokumentti. <https://infinum.co/the-capsized-eight/web-components-building-blocks-of-the-future-web>. 27.3.2014. Luettu 27.1.2017.

6 Nations, Daniel. 2016. Internet History – A Brief Look at the Key Events in Internet History. Verkkodokumentti. <https://www.lifewire.com/internet-history-3486216>. 7.10.2016. Luettu 20.1.2017.

7 Borodescu, Ciprian. 2013. Web Sites vs. Web Apps: What the Experts Think. Verkkodokumentti. <https://www.visionmobile.com/blog/2013/07/web-sites-vs-web-apps-what-the-experts-think>. 29.7.2013. Luettu 20.1.2017.

8 Calder, Jack. 2016. HTML5 vs Native Apps: What's Best for 2016? Verkkodokumentti. <http://justcreative.com/2016/03/17/html5-vs-native-apps-whats-best-for-2016/>. 17.3.2016. Luettu 21.1.2017.

9 Wodehouse, Carey. Front-End Web Development: Client-Side Scripting & User Experience. Verkkodokumentti. <https://www.upwork.com/hiring/development/how-scripting-languages-work/>. Luettu 21.1.2017.

10 Jana, Abhijit. 2010. What is the Difference Between Web Farm and Web Garden? Verkkodokumentti. <https://abhijitjana.net/2010/10/01/what-is-the-difference-between-web-farm-and-web-garden/>. 10.10.2010. Luettu 21.1.2017.

11 Clavijo, Diaz. 2013. Server-Side Programming Language Statistics. Verkkodokumentti. <http://blog.websitesframeworks.com/2013/03/programming-language-statistics-in-server-side-161/>. 6.3.2013. Luettu 21.1.2017.

12 ASP.NET. 2017. Verkkodokumentti. TechTerms. <https://techterms.com/definition/aspnet>. Luettu 25.2.2017.

13 Wilson, Dan. What is ColdFusion? Verkkodokumentti. Learn CF in a Week. <http://www.learncfinaweek.com/week1/What_is_ColdFusion_/>. Luettu 25.2.2017.

14 Singh, Chaitanya. 2015. Introduction to Java Programming. Verkkodokumentti. <http://beginnersbook.com/2013/05/java-introduction/>. 5.10.2015. Luettu 25.2.2017.

15 What is PHP? 2017. Verkkodokumentti. The PHP Group. <http://php.net/manual/en/intro-whatis.php>. Luettu 25.2.2017.

16 CSS Tutorial. 2017. Verkkodokumentti. W3Schools. <https://www.w3schools.com/css/>. Luettu 25.2.2017.

17 Getting Started with Flash Development. 2010. Verkkodokumentti. I Programmer. <http://www.i-programmer.info/programming/other-languages/1604-getting-started-with-flash-development.html>. 23.11.2010. Luettu 25.2.2017.

18 HTML5 Tutorial. 2017. Verkkodokumentti. W3Schools. <https://www.w3schools.com/html/>. Luettu 25.2.2017.

19 Rowinski, Dan. 2016. It's Official: JavaScript is the Most Commonly Used Programming Language on Earth. Verkkodokumentti. <https://arc.applause.com/2016/03/22/javascript-is-the-worlds-dominant-programming-language/>. 22.3.2016. Luettu 21.1.2017.

20 Houwens, Byron. 2016. An Introduction to TypeScript: Static Typing for the Web. Verkkodokumentti. <https://www.sitepoint.com/introduction-to-typescript/>. 20.6.2016. Luettu 25.2.2017.

21 JavaScript Libraries GitHub Search. 2017. Verkkodokumentti. GitHub. <https://github.com/search?utf8=%E2%9C%93&q=javascript+library&type=Repositories&ref=searchresults>. 6.3.2017. Luettu 6.3.2017.

22 JavaScript Frameworks GitHub Search. 2017. Verkkodokumentti. GitHub. <https://github.com/search?utf8=%E2%9C%93&q=javascript+framework&type=Repositories&ref=searchresults>. 6.3.2017. Luettu 6.3.2017.

23 The Definitive Source of the Best JavaScript Libraries, Frameworks, and Plugins. 2017. Verkkodokumentti. JavaScripting. <https://www.javascripting.com/>. Luettu 25.2.2017.

24 Barret, Lucy. 2016. Top JavaScript Frameworks for Building Web Applications. Verkkodokumentti. <https://www.templatemonster.com/blog/top-javascript-frameworks-building-web-applications/>. Luettu 25.2.2017.

25 Williams, Owen. 2016. The Most Popular JavaScript Library, jQuery, is Now 10 Years Old. Verkkodokumentti. <https://thenextweb.com/dd/2016/01/14/the-most-popular-javascript-library-jquery-is-now-10-years-old/#.tnw_tUGjkZxk>. 14.1.2016. Luettu 25.2.2017.

26 jQuery Introduction. 2017. Verkkodokumentti. W3Schools. <https://www.w3schools.com/jquery/jquery_intro.asp>. Luettu 25.2.2017.

27 Rathwell, Kaitlin. 2015. Web Components. Verkkodokumentti. <http://kaytcat.github.io/web-components/>. 28.8.2015. Luettu 23.1.2017.

28 Smith, Hedley. 2015. Thinking in Components. Verkkodokumentti. <http://www.hedleysmith.com/posts/thinking_in_components.html>. 6.8.2015. Luettu 23.1.2017.

29 Osmani, Addy. 2017. The PRPL Pattern. Verkkodokumentti. <https://developers.google.com/web/fundamentals/performance/prpl-pattern/>. 13.2.2017. Luettu 6.3.2017.

30 Creating Custom Directives. 2017. Verkkodokumentti. AngularJS. <https://docs.angularjs.org/guide/directive>. Luettu 23.1.2017.

31 Jordan, Todd. 2015. A Simplified View of MV* Web Pattern Differences. Verkkodokumentti. <http://presentationtier.com/a-simplified-view-of-mv-web-pattern-differences/>. 26.2.2015. Luettu 23.1.2017.

32 Getting Started with Web Components and PolymerJS. 2016. Verkkodokumentti. Code Teddy. <https://codeteddy.com/2016/06/21/getting-started-with-web-components-and-polymerjs/>. Luettu 12.2.2017.

33 Bidelman, Eric. 2013. HTML's New Template Tag. Verkkodokumentti. <https://www.html5rocks.com/en/tutorials/webcomponents/template/>. 26.2.2013. Luettu 12.2.2017.

34 Motto, Todd. 2014. Web Components and Concepts, ShadowDOM, Imports, Templates, Custom Elements. Verkkodokumentti. <https://toddmotto.com/web-components-concepts-shadow-dom-imports-templates-custom-elements/>. 2.7.2014. Luettu 12.2.2017.

35 Bidelman, Eric. 2017. Custom Elements v1: Reusable Web Components. Verkkodokumentti. <https://developers.google.com/web/fundamentals/getting-started/primers/customelements>. 9.2.2017. Luettu 12.2.2017.

36 The HTML DOM (Document Object Model). Verkkodokumentti. W3Schools. <https://www.w3schools.com/js/js_htmldom.asp>. Luettu 12.2.2017.

37 Bidelman, Eric. 2017. Shadow DOM v1: Self-Contained Web Components. Verkkodokumentti. <https://developers.google.com/web/fundamentals/getting-started/primers/shadowdom>. 9.2.2017. Luettu 12.2.2017.

38 Rimmer, Jon. 2017. Are We Componentized Yet? Verkkodokumentti. <http://jonrimmer.github.io/are-we-componentized-yet/>. Luettu 13.2.2017.

39 Libraries. Verkkodokumentti. Webcomponents.org. <https://www.webcomponents.org/libraries>. Luettu 13.2.2017.

40 Precht, Pascal. 2014. Polymer vs. X-Tag – Here's the Difference. Verkkodokumentti. <https://pascalprecht.github.io/2014/07/21/polymer-vs-x-tag-here-is-the-difference/>. 21.7.2014. Luettu 13.2.2017.

41 Hernandez, Alejandro. 2014. Polymer.js: The Future of Web Application Development? Verkkodokumentti. <https://www.toptal.com/front-end/polymer-js-the-future-of-web-application-development>. Luettu 13.2.2017.

42 Polymer Library. 2016. Verkkodokumentti. Polymer Project. <https://www.polymer-project.org/1.0/docs/devguide/feature-overview>. Luettu 26.2.2017.

43 Registration and Lifecycle. 2016. Verkkodokumentti. Polymer Project. <https://www.polymer-project.org/1.0/docs/devguide/registering-elements>. Luettu 26.2.2017.

44 Local DOM Basics and API. 2016. Verkkodokumentti. Polymer Project. <https://www.polymer-project.org/1.0/docs/devguide/local-dom>. Luettu 26.2.2017.

45 Miles, Scott. 2015. What is Shady DOM? Verkkodokumentti. <https://www.polymer-project.org/blog/shadydom>. 28.5.2015. Luettu 26.2.2017.

46 X-Tag/X-Tag. 2017. Verkkodokumentti. GitHub. <https://github.com/x-tag/x-tag>. 9.3.2017. Luettu 9.3.2017.

47 Polymer/Polymer. 2017. Verkkodokumentti. GitHub. <https://github.com/Polymer/polymer>. 9.3.2017. Luettu 9.3.2017.

48 X-Tag. 2017. Verkkodokumentti. Stack Overflow. <http://stackoverflow.com/tags/x-tag/info>. 9.3.2017. Luettu 9.3.2017.

49 Polymer. 2017. Verkkodokumentti. Stack Overflow. <http://stackoverflow.com/tags/polymer/info>. 9.3.2017. Luettu 9.3.2017.

50 X-Tag Builds. 2017. Verkkodokumentti. X-Tag. <http://x-tag.github.io/builds>. Luettu 9.3.2017.

51 Parashar, Pankaj. Building a <google-map> Custom Element with X-Tag. Verkkodokumentti. Codepen. <http://codepen.io/SitePoint/pen/VevVpa/>. Luettu 9.3.2017.

52 Google Maps Embed API. 2017. Verkkodokumentti. Google Maps APIs. <https://developers.google.com/maps/documentation/embed/guide#embedding_the_map>. Päivitetty 15.3.2017. Luettu 9.3.2017.

53 X-Tag Docs. Verkkodokumentti. X-Tag. <http://x-tag.github.io/docs>. Luettu 9.3.2017.

54 Bower: A Package Manager for the Web. Verkkodokumentti. Bower. <https://bower.io/>. Luettu 9.3.2017.

55 Polymer CLI. 2016. Verkkodokumentti. Polymer Project. <https://www.polymer-project.org/1.0/docs/tools/polymer-cli>. Luettu 12.3.2017.

56 Elements: a Collection of Elements Made by the Polymer Team. Verkkodokumentti. Webcomponents.org. <https://www.webcomponents.org/collection/Polymer/elements>. Luettu 15.3.2017.

57 Polymer – Polymer. 2017. Verkkodokumentti. Gemnasium. <https://gemnasium.com/bower/polymer-polymer/versions>. 19.3.2017. Luettu 19.3.2017.

58 Rouse, Margaret. 2016. RESTful API. Verkkodokumentti. <http://searchcloudstorage.techtarget.com/definition/RESTful-API>. 12/2016. Luettu 19.3.2017.

59 Foursquare. 2017. Verkkodokumentti. Foursquare. <https://foursquare.com/>. Luettu 19.3.2017.

60 Iron-ajax. 2016. Verkkodokumentti. Webcomponents.org. <https://www.webcomponents.org/element/PolymerElements/iron-ajax/iron-ajax>. 5/2016. Luettu 19.3.2017.

61 Dom-repeat. 2016. Verkkodokumentti. Polymer Project. <https://www.polymer-project.org/1.0/docs/api/dom-repeat>. Luettu 19.3.2017.

62 App-route. 2016. Verkkodokumentti. Webcomponents.org. <https://www.webcomponents.org/element/PolymerElements/app-route>. Luettu 19.3.2017.

63 Iron-pages. 2016. Verkkodokumentti. Webcomponents.org. <https://www.webcomponents.org/element/PolymerElements/iron-pages>. 5/2016. Luettu 19.3.2017.

64 Google-map. 2016. Verkkodokumentti. Webcomponents.org. <https://www.webcomponents.org/element/GoogleWebComponents/google-map/google-map>. 5/2016. Luettu 19.3.2017.

65 Geo-location. 2016. Verkkodokumentti. Webcomponents.org. <https://www.webcomponents.org/element/ebidel/geo-location>. 5/2016. Luettu 19.3.2017.

66 Wagner, Janet. 2015. Top 10 Mapping APIs: Google Maps, Microsoft Bing Maps and MapQuest. Verkkodokumentti. <https://www.programmableweb.com/news/top-10-mapping-apis-google-maps-microsoft-bing-maps-and-mapquest/analysis/2015/02/23>. 23.2.2015. Luettu 19.3.2017.

67 Campbell, Steve. 2010. 5 Best Location-Based Alternatives to Foursquare. Verkkodokumentti. <http://www.makeuseof.com/tag/5-locationbased-alternatives-foursquare/>. 27.7.2010. Luettu 20.3.2017.